PATENT
Attorney Docket No.: 020375-007400US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:

Larry C. Frame et al.

Application No.: 10/044,484

Filed: January 11, 2002

For: Methods And Systems For Extracting Related Information From Flat Files

Customer No.: 20350

Confirmation No.: 9883

Examiner: Debbie M. Le

Art Unit: 2168

DECLARATION PURSUANT TO 37 C.F.R. § 1.131

Mail Stop **AF**
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

I, Larry C. Frame, declare as follows:

1. I am a co-inventor of the subject matter of the above-referenced patent application.

2. Prior to November 21, 2001, I participated in reducing to practice the subject matter of the patent application (hereinafter "the invention") as described in Claim 1, namely,

a) in response to a user input that designates at least one field as a key segment, wherein a key segment comprises a field having pre-populated data and wherein the key segment field is common to each of a plurality of

the records, comparing data contained in the key segment of each record of a first file to data in a related key segment of each record of a second file;

b) upon each occurrence of a match of data in the key segment of a record in the first file to data in the related key segment of a record in the second file, creating a record in a temporary electronic file, wherein the record in the temporary file includes at least one field and wherein the at least one field includes a copy of the matching data from the;

c) selecting data from the temporary file; and

d) outputting the selected data.

3. Submitted herewith in support thereof is a source code file (Exhibit A). Page 3 (user specified request) of Exhibit A correlates to clause a of Claim 1. Page 28 (selecting like keyed records) of Exhibit A correlates to clause b of Claim 1. Pages 23 and 25 (selecting data) of Exhibit A correlate to clause c of Claim 1. Pages 8 (final outputting) and 23 and 24 (temporary outputting) of Exhibit A correlate to clause d of Claim 1. The Change Log on page of Exhibit A shows that the invention had been reduced to practice by at least June 6, 2001.

4. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and, further, that these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the above-referenced application or a patent issued therefrom.

Respectfully submitted,

Date: 2005/12/29

Larry C. Frame

```
/* REXX */    /* lcframe - 05/16/01 */
/*### NOTE: this version is the next step after version DVSQL4 that has ###*/
/*### the initial try at AND/OR logic for processing a single SELECT    ###*/
/*### alias output reference.  Also this includes the changes to do all ###*/
/*### the WHERE record selection as a group procedure effort and the    ###*/
/*### associated SELECT options/reformatting will process as a procedure###*/
/*### effort AFTER all WHERE selection processing has finished.         ###*/
/*------------- DVSQL (DV's version3 of SQL for flat files) -----------------*\
|  SQL processor for inquiries involving non-DB2 files.                       |
|-----------------------------------------------------------------------------|
|  The following are processing verbs currently available for this processor  |
|    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -    |
|  SELECT  - definition of selected data to be used as output for the query    |
|      format: SELECT {DISTINCT} {sub-parms,} field1, field2, etc.            |
|      Sub Parameters:                                                        |
|             DISTINCT - per the selected fields, make the list singular      |
|             COUNT    - number of records selected                           |
|                 format: COUNT({DISTINCT} field)                             |
|             MAX      - largest value for the specified field               |
|                 format: MAX(field)                                          |
|             MIN      - smallest value for the specified field              |
|                 format: MIN(field)                                          |
|      field format: file-letter.(displacement,length)                        |
|             where file-letter is the alphabetic letter associated with the  |
|             input DDname on the FROM statement and "displacement" and       |
|             "length" describe the location of the field in the input record.|
|                              - or -                                         |
|      field format: 'literal-value'                                          |
|             where literal value is any character/s that are to be inserted  |
|             into the output record.                                         |
|  FROM    - input DDnames and alias letter (maximum of 2 per SELECT)         |
|     format: FROM DDname1 file-letter1, DDname2 file-letter2                 |
|         where "DDname" is a DD/filename defined in the JCL of the JOB and   |
|         "file-letter" is an UPPER-CASE alphabet letter to be used as a      |
|         short-hand association to the file when describing field name for   |
|         use with other verbs.                                               |
|  INTO    - output DDname (default is SYSOUT or work file/table)             |
|       format: INTO DDname                                                   |
|  WHERE   - conditions of processing                                         |
|     format 1: WHERE A = B        =,<>,<,<=,>,>=  compare                    |
|     format 2: WHERE A IN C       select A values that are in list B         |
|     format 3: WHERE A NOT IN C   select A values that are not in list B     |
|     format 4: WHERE A NUMERIC    class tests  ALPHA, INTERGER, ALPHANUMERIC |
|     format 5: WHERE cond AND cond    intersection of condition outputs       |
|     format 6: WHERE cond OR cond     union of condition outputs              |
|     format 7: WHERE A BETWEEN value1 AND value2                             |
|        NOTE: in the above formats, A designates an input field,             |
|            B designates an input field or a literal value ( 'xxxxxx' ),     |
|            C designates either a user defined table of values               |
|              "('A','B','C','etc')" or a sub-query "(SELECT etc etc)"        |
|  ORDER BY  - sorted order to save output in                                 |
|      format: ORDER BY field1, field2, field3, etc.     where...            |
|          fieldx is  (displacement,length){ order}                           |
|              displacement - location of field in output record              |
|              length       - length of field at specified location           |
|              order        - ASC (ascending - default) or DESC (descending)   |
|-----------------------------------------------------------------------------|
| Note: When using the NOT IN option of the WHERE verb, it is assumed that    |
|   the user will not be selecting any fields from the control file for use   |
|   in the output, as that would be stupid since you are looking for compare  |
|   records that don't have their key in the control file.                    |
|-----------------------------------------------------------------------------|
| Note: This processor processes ALL logical file relations and comparisons   |
|   (WHERE information) associated with an SQL level first, then does ALL     |
|   SELECT record reformatting on the resulting file of information.          |
|       A WHERE comparison involving more than one file (a compare and control|
|   file) results in a compound record structure consisting of selected       |
|   comparison-file records suffixed with the logically paired control-file   |
|   records.  This causes the DVSQL program to use enhanced field referencing |
|   when doing the final SELECT processing for each SQL level since the      |
|   requested field may reside in the suffixed (extended) portion of the      |
|   WHERE output record.                                                      |
|       If complex (multi-file) compare is to be done for any SQL level, it   |
|   must be the first compare in the WHERE verb for that level.  This tells   |
|   the processor that the rest of the comparisons of that level will involve |
|   use of a compound file structure.                                         |
|-----------------------------------------------------------------------------|
|  Control cards are inputted via the SYSIN DD.                               |
```

```
\*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*\
|  Passed parm information is as follows:                                   |
|      PNODE  -  The primary node to use in creating sort work data sets and |
|          other needed work files.                                         |
|      WKDISP -  Valid values are KEEP and DELETE (<--default). This tells  |
|          the processor whether to keep or delete generated work files     |
|      OUTDSN -  data set name of the default output file to be automatically |
|          generated in place of specifying the INTO verb.  If OUTDSN is    |
|          specified, the INTO SQL verb will be ignored.                    |
\*-------------------------------------------------------------------------*/
  parse upper arg PNODE WKDISP OUTDSN JUNK

/*-------------------------------------------------------------------------*\
|*  Make sure a Primary NODE value was specified                           |
\*-------------------------------------------------------------------------*/
  if PNODE = '' | PNODE = 'HELP'    then
      do
        say ' '
        say ' '
        say '  Format of //SYSTSIN DD * control card is as follows:'
        say '       " %DVSQL PNODE WKDISP OUTDSN "'
        say '  where  PNODE is the primary node to catalog all work areas and'
        say '               data sets (mandatory field)'
        say '         WKDISP is the disposition of all work data sets used and'
        say '               created in the DVSQL process.  Valid values are '
        say '               *, KEEP, DELETE.  KEEP causes all work data sets to'
        say '               be kept after processing is complete.  DELETE or *'
        say '               (the defaults) cause all work data sets to be '
        say '               deleted (cleaned up) after processing is complete.'
        say '         OUTDSN is the name of the output data set to be used to'
        say '               store the DVSQL output.  This option overrides any'
        say '               use of the INTO verb in the SQL requests.'
        say ' '
        say ' '
        say '     ++ Options and Format of DVSQL Statements ++'
        say ' '
        call SELECT_FORMAT
        call FROM_FORMAT
        call INTO_FORMAT
        call WHERE_FORMAT
        call ORDER_BY_FORMAT
        exit 8
      end

/*-------------------------------------------------------------------------*\
|*  Verify inputted WKDISK parm                                            |
\*-------------------------------------------------------------------------*/
  select
      when WKDISP = '*'       then WKDISP = 'DELETE'    /* use default */
      when WKDISP = 'DELETE'  then nop
      when WKDISP = 'KEEP'    then nop
      when WKDISP = ''        then WKDISP = 'DELETE'    /* use default */
      otherwise
          do
            say '*************'
            say '**  ERROR  **'
            say '*************'
            say '   The inputted work data set disposition PARM value',
                '"'WKDISP'" is invalid.'
            say '   Valid values are:'
            say '        "DELETE" - delete all work data sets generated'
            say '        "KEEP"   - keep all generated work data set'
            say '        "*"      - use the DELETE default'
            return 8
          end
  end

  say ' '
  say '---- Specified DVSQL command line PARMs ----'
  say '          PNODE  = 'PNODE
  say '          WKDISP = 'WKDISP
  if OUTDSN = ''    then
      say '          OUTDSN = N/A'
  else
      say '          OUTDSN = 'OUTDSN
  say ' '
```

```
/*--------------------------------------------------------------------*\
|*  Put the user inputted SQL control cards onto an internal SYSIN. table   |
\*--------------------------------------------------------------------*/
  x = outtrap('DUMMY.')
 "alloc f(SYSIN)"
  x = outtrap('OFF')
 "execio * diskr SYSIN (stem SYSIN. finis"
  REXX_RC = RC
 "free f(SYSIN)"
  if REXX_RC > 0    then
      do
        say ' Error reading SYSIN, execio RC = 'REXX_RC
        return 16
      end
```

USER SPECIFIED REQUEST – a)

The red highlighted code below (and the routines called from it) reviews the user provided control card request and generates the tokens necessary to process the request through the relational model utility. The following tokens are generated for processing:

```
#### list of level 1 DVSQL tokens ####
    OUTFLD IN1.,5,45
    OUTFLD IN2.,1,39
    OUTFLD IN2.,50,25
    DISKR IN00IN1 IN1
    DISKR IN00IN2 IN2
    DISKW SQLOUT
    EQUATE IN1.,1,4
    EQUATE EQ
    EQUATE IN2.,40,4
```

- The two DISKR tokens identify the two input files that need to be allocated.
- The DISKW token identifies the final output file that needs to be allocated.
- The three EQUATE tokens set up the relation condition in which a key field of records in file IN1 that starts in position 1 for a length of four is EQUAL to a key field of records in file IN2 that starts in position 40 for a length of 4.
- The three OUTFLD tokens, identify, in order, the data fields of file IN1 and IN2 that are to be used as output. So, when the above condition of key fields is satisfied, a record will be written to output that contains the data from the like keyed record in file IN1 that starts in byte 5 for a length of 45, followed by the data in the like keyed record in file IN2 that starts in byte 1 for a length of 39 and in byte 50 for a length of 25.

```
/*--------------------------------------------------------------------*\
|*    do compiler stuff to verify format and content of control cards       |
|*    and generate a stack of processing tokens                             |
\*--------------------------------------------------------------------*/
  QUOT = "'"
  DDS = 0           /* init general query DD list counter */
  OUT#DD = 'SQLOUT' /* set output default, in case there is no INTO verb */
  OUT#DSN = ''      /* initialize DSN relative to INTO verb */
  SORT#CARD = ''    /* initialize area to store parsed ORDER BY information */
  I = 1             /* set current query level */
  IMAX = 1          /* set current highest query level */
  call INIT_NEW_QUERY_LEVEL_FIELDS
  say ' '
  say '                       DVSQL Compile Parsing Messages'
  say '     ----+----1----+----2----+----3----+----4----+----5----+----6----+--'
  IN_CNT = 0
  UTBL_CNT = 0          /* for tracking internal WHERE IN user tables */
  PARS_LINE = ''
  do forever
    parse upper var PARS_LINE VERB PARS_LINE
    if VERB = ''    then
        do
          call READ_SYSIN
          iterate
        end
    say 'VERB = *'VERB'*'
    select
      when VERB = 'SELECT'     then
        call SELECT_VERB
      when VERB = 'FROM'       then
```

```
        call FROM_VERB
      when VERB = 'INTO'        then
        call INTO_VERB
      when VERB = 'WHERE'       then
        call WHERE_VERB
      when VERB = 'ORDER'     then
        call ORDER_BY_VERB
      otherwise
        do
          say ' *'VERB'* in the current line is not recognized'
          return 8
        end
    end
    select
      when substr(PARS_LINE,1,1) = ')'     then    /* end-of-query-level */
          do
          do while substr(PARS_LINE,1,1) = ')'
            call END_OF_LEVEL_CHECKS
            call SAVE_QUERY_FIELD_COUNTERS    /* save current Jx counters */
            I = CAME_FROM.I                   /* reset prior query level */
            call RESTORE_QUERY_FIELD_COUNTERS /* restore current Jx counters */
            PARS_LINE = substr(PARS_LINE,2)   /* parse off down-level delim */
            PARS_LINE = strip(PARS_LINE,'l')  /* drop leading blanks  */
          end                                 /* where WHERE left off... */
          PARS_LINE = 'WHERE 'PARS_LINE  /* force back into WHERE processing */
          end
      when substr(PARS_LINE,1,1) = '('     then
          do
            call SAVE_QUERY_FIELD_COUNTERS    /* save current Jx counters */
            IMAX = IMAX + 1                   /* set next available level */
            CAME_FROM.IMAX = I                /* set level return counter */
            I = IMAX                          /* set new current level */
            call INIT_NEW_QUERY_LEVEL_FIELDS  /* init new bucket counters */
            PARS_LINE = substr(PARS_LINE,2)   /* parse off up-level delim */
          end
      otherwise nop
    end
  end
END_COMPILER:
  call END_OF_LEVEL_CHECKS
  call SAVE_QUERY_FIELD_COUNTERS     /* save current Jx counters */

/*---- print each level's stack of tokens prior to processing ----*/
  say ' '
  say ' '
  say ' '
  do I = 1 to IMAX
    say '#### list of level 'I' DVSQL tokens ####'
    interpret "J = SELECT#"I".0"
    do II = 1 to J
      interpret "say '    OUTFLD 'SELECT#"I".II"
    end
    interpret "J = FROM#"I".0"
    do II = 1 to J
      interpret "say '    DISKR 'FROM#"I".II"
    end
    if I = 1  &  OUT#DD <> ''     then
        say '    DISKW 'OUT#DD
    interpret "J = WHERE#"I".0"
    if J > 0    then
        do II = 1 to J
          interpret "say '    EQUATE 'WHERE#"I".II"
        end
    if I = 1  &  SORT#CARD <> ''     then
        say '  'SORT#CARD
  end

/*-------------------------------------------------------------------------*\
|          -  Process each stack of generated tokens...  -                 *|
|  Each stack represents a layer of processing to be performed. Processing *|
|  starts from the bottom stack (last layer of SQL code compiled) and      *|
|  works its way to the top.                                               *|
\*-------------------------------------------------------------------------*/
  say ' '
  say ' '
  say ' '
  SORT_CNT = 0       /* initialize counter to aid in generating sort outputs */
  WORK_CNT = 0       /* initialize counter to aid generating compare outputs */
```

```
WKDSN.0 = 0          /* initialize work data set list */
WORKFILE = ''        /* initialize inter-step processing results file */
do I = IMAX to 1 by -1       /* process SQL stacks from bottom to top */
  drop IN_FILE. DD. DSN.     /* clear off prior input file info */
  drop OUT#FLD.      /* clear off prior output field information */
  OUT#FLD.0 = 0      /* initialize output fields counter */
  WHERE_DATA = ''    /* set WHERE verb existance field */
  DISTINCT = ''      /* initialize DISTINCT function */
  SEL_OPT = ''       /* initialize variable to hold MIN, MAX, etc option */
  say ' '
  say ' ---- SQL (level 'I') ---- Token Diagnostics'
  /*-------------------------------------------------------------------*\
  |   step thru tokens in the current stack to set up processing options   |
  \*-------------------------------------------------------------------*/
  /*-- set current level SELECT options --*/
  interpret "JS = SELECT#"I".0"      /* SELECT field count for level */
  do J = 1 to JS
    interpret "DATA = SELECT#"I".J"
    select
      when DATA = 'AVG'          then
          SEL_OPT = 'AVG'
      when DATA = 'COUNT'        then
          SEL_OPT = 'COUNT'
      when DATA = 'DISTINCT'     then
          DISTINCT = 'ON'
      when DATA = 'MAX'          then
          SEL_OPT = 'MAX'
      when DATA = 'MIN'          then
          SEL_OPT = 'MIN'
      when DATA = 'SUM'          then
          SEL_OPT = 'SUM'
      otherwise
          do
            OUT#FLD.0 = OUT#FLD.0 + 1
            interpret "OUT#FLD."OUT#FLD.0"=DATA"
          end
    end
  end

  /*-- set current level INTO options --*/
  /*   The OUT#DD variable hold the information from compile time.     */
  /*   Since it can only exist for the primary SQL level, there was no */
  /*   need to hold the data in a table ------------------------------*/

  /*-- set current level WHERE information --*/
  interpret "JW = WHERE#"I".0"          /* WHERE field count for level */
  /*-- check to see if the first compare is complex (involves two --*/
  /*-- different files) if so, flag as a complex file WHERE request */
  COMPOUND = ''        /* initialize output record type flag */
  /*-- DDALIAS.1 and .2 identify the two main files (if both) being used --*/
  /*-- in the SQL level.  If a compound file structure is generated as the */
  /*-- result of a multi-file compare with an EQ or NE operator, DDALIAS.1 */
  /*-- will identify the files involved and the ordering of the information*/
  /*-- in the structure.  ------------------------------------------------*/
  DDALIAS.1 = ''  /* initialize hold area for primary compare file alias */
  DDALIAS.2 = ''  /* initialize hold area for primary control file alias */
  if JW > 0    then   /* if there is where data... */
      do
        interpret "parse var WHERE#"I".1 DDALIAS1 ',' JUNK"
        DDALIAS.1 = strip(DDALIAS1,'t','.')  /* set the compare file char */
        if JW > 2    then
            do
              interpret "OPER_CHK = WHERE#"I".2"
              if OPER_CHK = 'EQ'  |  OPER_CHK = 'NE'    then
                do /*-- possible complex file compare --*/
                  interpret "parse var WHERE#"I".3 DDALIAS2 ',' JUNK"
                  if datatype(substr(DDALIAS2,1,1),'U') = 1   then
                      if DDALIAS1 = DDALIAS2  then nop
                      else     /* 1st compare involves two different files */
                          DDALIAS.2 = strip(DDALIAS2,'t','.')
                end
            end
      end
  /*-- concatenate all WHERE data into a single function string --*/
  do J = 1 to JW
    interpret "DATA = WHERE#"I".J"
    if substr(DATA,1,7) = 'SUBQRY#'    then     /* if prior subquery */
        do                          /* results are being used as input... */
```

```
        SQ_NUM = substr(DATA,8)              /* obtain subquery number */
        interpret "JF = FROM#"I".0"
        JF = JF + 1
        interpret "FROM#"I".0 = JF"    /* add dummy DDNAME and alias to */
        interpret "FROM#"I".JF = "DATA" SQ#"SQ_NUM  /* current FROM list*/
      end
  WHERE_DATA = WHERE_DATA||strip(DATA,t)' '
end

/*-- set current level FROM information --*/
DDLRECL.1 = 0    /* initialize value for use in compound record formats */
DDLRECL.2 = 0    /* initialize value for use in compound record formats */
WORK_LRECL = 0   /* initialize LRECL for WHERE compare results */
interpret "JF = FROM#"I".0"        /* FROM file count for level */
IN_FILE.0 = JF                     /* initialize FROM input file counter */
do J = 1 to JF
  interpret "DDNAME_DDALIAS = FROM#"I".J"
  parse var DDNAME_DDALIAS DDNAME DDALIAS
  interpret "DD."DDALIAS" = '"DDNAME"'" /* set relative DD */
  IN_FILE.0 = IN_FILE.0 + 1            /* increase input file count */
  IN_FILE.J = DDALIAS
  /* set other relative DD info for possible later needs */
  if substr(DDALIAS,1,3) = 'SQ#'   then /* for subquery output files... */
      do
        interpret "SQ_DSN = "DDNAME
        interpret "DSN."DDALIAS" = SQ_DSN"  /* set relative subquery DSN */
        say '      - DSN        = 'SQ_DSN
      end
  else                                 /* for standard FROM input files */
      do
        x = listdsi("'"DDNAME"' file")
        if SYSDSORG = 'PO'    then  /* resolve DSN member name */
            do
              x = outtrap('STUFF.')
             "listalc status"
              x = outtrap('OFF')
              do II = 1 to STUFF.0
                if substr(STUFF.II,1,10) = '  'left(DDNAME,8)    then
                    do
                      II = II - 1
                      AA = length(SYSDSNAME)
                      if substr(STUFF.II,1,AA) = SYSDSNAME    then
                          do
                            MBRNAME = substr(STUFF.II,AA+2)
                            parse var MBRNAME MBRNAME ')'
                            say '      - MBRNAME    = 'MBRNAME
interpret "DSN."DDALIAS"='"SYSDSNAME"("MBRNAME")'" /* set relative DSN */
                            leave
                          end
                      else
                          do
                       say '   unable to resovle DSN member name for 'DDNAME
                            call FROM_ERROR
                          end
                    end
              end
              if II > STUFF.0    then
                  do
                    say '   unable to resovle DSN member name for 'DDNAME
                    call FROM_ERROR
                  end
              drop STUFF.
            end
        else
            interpret "DSN."DDALIAS" = '"SYSDSNAME"'" /* set relative DSN */
        say '      - DSN        = 'SYSDSNAME
        if SYSREASON > 0    then
            if SYSREASON = 12    then /* VSAM data sets not supported */
              do
               "alloc f(SORTIN) da('"SYSDSNAME"') SHR"
                LRECLCHK = PNODE".LRECLCHK.S"time('S')
                x = outtrap("DUMMY.")
               "DELETE '"LRECLCHK"'" /* cleanup up possible prior version */
                x = outtrap("OFF")
               "alloc f(SORTOUT) da('"LRECLCHK"') new delete " ,
                   " unit(SYSDA) space(1,1) tracks " ,
                   " dsorg(PS) recfm(F,B) blksize(0)"
               "alloc f(SYSIN) da(SYSIN) unit(SYSDA) space(1,1) tracks "  ,
```

```
                    " dsorg(PS) recfm(F,B) lrecl(80) blksize(80) new delete"
                  LCHK.0 = 1
                  LCHK.1 = '  SORT FIELDS=COPY,STOPAFT=1 '
                 "execio 1 diskw SYSIN (stem LCHK. finis"
                 "alloc f(SYSOUT) DUMMY"
                  /* "call 'FDR.SYNCR36.LINKLIB(SYNCSORT)'"  */
                  address ATTCHMVS "SORT"
                  /* use LRECL of first record */
                  if RC = 0    then
                      x = listdsi("'"LRECLCHK"'")
                  else
                      say '**WARNING** could not obtain LRECL info for ',
                          'the data set 'SYSDSNAME
                 "free f(SORTIN SORTOUT SYSIN SYSOUT)"
                 end
              else
                  do
                    say '      **** WARNING ****'
                    say '      *  Problems obtaining file characteristics',
                        'for input DD 'DDNAME
                    say '      *  Default characteristics will be applied'
                    say '      *      REASON = 'SYSREASON
                    say '      *      MSG    = 'SYSMSGLVL1
                    say '      *****************'
                  end
          if DDALIAS = DDALIAS.1  then
              DDLRECL.1 = SYSLRECL
          else
          if DDALIAS = DDALIAS.2  then
              DDLRECL.2 = SYSLRECL
        end
    end
    if JW = 0    then  /* if no WHERE verb info , */
        do      /* provide a filename for SELECT processing to use for input */
say '======= no WHERE data, set WORKFILE for SELECT processing ======'
        WORK_LRECL = 0
        if OUT#FLD.1 = '*'    then
            do
say '======= the SELECT ALL option was specified'
            WORKFILE = SYSDSNAME  /* for "SELECT ALL", use default DSN */
say '======= WORKFILE = 'WORKFILE
            end
        else    /* otherwise,  */
            do  /* obtain DSN from 1st alias found in SELECT out#flds */
              do II = 1 to JS
                if substr(OUT#FLD.II,1,1) = "'"    then nop
                else
                    do
                      parse var OUT#FLD.II DDALIAS '.,' JUNK
say '======= OUT#FLD.'II'='OUT#FLD.II'    DDALIAS='DDALIAS
                      leave
                    end
              end
              interpret "WORKFILE = DSN."DDALIAS
say '======= WORKFILE = 'WORKFILE
            end
        end
    else
        /*-- if the initial WHERE compare indicated generation of a compound */
        /*-- record structure, add the LRECLs of both files to get the work -*/
        /*-- file LRECL.  Otherwise, just use the "compare" file LRECL ------*/
        if DDALIAS.2 = ''    then
            WORK_LRECL = DDLRECL.1              /* set standard LRECL */
        else
            WORK_LRECL = DDLRECL.1 + DDLRECL.2  /* set compound LRECL */

    /*-- set current level ORDERED BY information --*/
    /*   The SORT#CARD variable holds the information from compile time. */
    /*   Since it can only exist for the primary SQL level, there was no */
    /*   need to hold the data in a table ----------------------------*/
    say ' '
    say ' ---- SQL (level 'I') ---- Processing Diagnostics'

    call PROCESS_SQL_LEVEL    /* process the current task level */

    interpret "XXXX = SELOPT#"I  /* check to see if output is function result */
    if XXXX > ''    then                /* if so, */
        interpret "SUBQRY#"I" = '¢'XXXX"  /* store result in subqry level var */
```

```
   else                                  /* otherwise, */
        interpret "SUBQRY#"I" = WORKFILE" /* store DSN of resulting set/table */
  end

  /*---- if the final result is a function output, store the result on the --*/
  /*---- last used work file beforfe doing the final file naming stuff ------*/
  if substr(SUBQRY#1,1,1) = '¢'    then
      do
       "alloc f(WORKFILE) da('"WORKFILE"') shr"
        queue substr(SUBQRY#1,2)
       "execio 1 diskw WORKFILE (finis"
       "free f(WORKFILE)"
      end

  /*---- processing of ORDER BY verb info to make final reording of data ----*/
  if SORT#CARD > ''    then
      do
        say ' '
        say ' -- ORDER BY option processing --'
        drop SORTCARD.
        SORTCARD.1 = SORT#CARD
        SORTCARD.0 = 1
        /*---- reorder the file as requested ----*/
        call SORT_DATA 'SETUP' WORKFILE
        WORKFILE = SORTOUT
      end
```

FINAL OUTPUTTING - d)

This segment of code directs the last used temporary output data set to final output destination via a few different methods. If no OUTDSN was specified by the user, the last temporary output file (WORKFILE) is allocated as input, the final output destination (OUT#DD) is allocated for output use, and the input is copied directly to the output device whether it be the default utility data set or the SYSOUT output writer. If an OUTDSN was specified, the last used temporary output file is simply renamed as per user specification to the indicated data set name.

```
  /*---- put results onto final output file ----*/
  if OUTDSN = ''    then
      do
       "alloc f(WORKFILE) da('"WORKFILE"') shr"
        x = outtrap('TRASH.')
       "alloc f("OUT#DD")"
        x = outtrap('OFF')
        if OUT#DD = 'SQLOUT'    then
            do
              say ' '
              say ' '
              say '>>>> Output is on SQLOUT <<<<'
            end
        else
            do
              x = listdsi("'"OUT#DD"' file")
              say ' '
              say ' '
              say '>>>> Output is on 'SYSDSNAME' <<<<'
            end
       "execio 0 diskw "OUT#DD" (open"
       "execio 1 diskr WORKFILE"
        do while RC = 0
         "execio 1 diskw "OUT#DD
         "execio 1 diskr WORKFILE"
        end
       "execio 0 diskr WORKFILE (finis"
       "execio 0 diskw "OUT#DD" (finis"
       "free f(WORKFILE "OUT#DD")"
        call DELETE_WORK_DATA_SETS
      end
  else
      do
        call IDCAMS_RENAME
        WKDSN.0 = WKDSN.0 - 1
        call DELETE_WORK_DATA_SETS
      end
  exit

INIT_NEW_QUERY_LEVEL_FIELDS:
/*-------------------------------------------------------------------------*\
```

```
|    initialize array fields for starting up a new query level              |
\*-------------------------------------------------------------------------*/
  JS = 0
  JF = 0
  JW = 0
  call SAVE_QUERY_FIELD_COUNTERS
  return

SAVE_QUERY_FIELD_COUNTERS:
/*-------------------------------------------------------------------------*\
|    save the array bucket counters for the current query level             |
\*-------------------------------------------------------------------------*/
  interpret "SELECT#"I".0 = JS"        /* SELECT fields */
  interpret "FROM#"I".0 = JF"          /* input DDs */
  interpret "WHERE#"I".0 = JW"         /* WHERE fields and operators */
  return

RESTORE_QUERY_FIELD_COUNTERS:
/*-------------------------------------------------------------------------*\
|    restore the array bucket counters for the current query level          |
\*-------------------------------------------------------------------------*/
  interpret "JS = SELECT#"I".0"
  interpret "JF = FROM#"I".0"
  interpret "JW = WHERE#"I".0"
  return

READ_SYSIN:
/*-------------------------------------------------------------------------*\
|    a generic read of the SYSIN. table generated from the user inputted    |
|    SQL control cards.                                                     |
\*-------------------------------------------------------------------------*/
  IN_CNT = IN_CNT + 1
  if IN_CNT > SYSIN.0    then
      do
        call SAVE_QUERY_FIELD_COUNTERS
        signal END_COMPILER
      end
  say ' **  'SYSIN.IN_CNT'  **'
  PARS_LINE = strip(SYSIN.IN_CNT,b)
  return

END_OF_LEVEL_CHECKS:
/*-------------------------------------------------------------------------*\
|    Checks to be done when the end of an SQL level is encountered.  There  |
|    are some minimum processing requirements that must be satisfied to be  |
|    able to process a request.                                             |
\*-------------------------------------------------------------------------*/
  /*-- SQL limitation... DVSQL will allow for more than 1 SELECT field in --*/
  /*-- sub-querys to allow for ease of generating tables without having   --*/
  /*-- unnecessary AND/OR logic in the higher level query ----------------*/
/*if I > 1    then
      if JS > 1    then
          do
            say '  Too many fields specified'
            say '  Sub-queries are limited to one SELECT field'
            call SELECT_ERROR
          end  */
  /*-- must have at least one field specified on the SELECT verb --*/
  if JS = 0    then
      do
        say '  Must have at least one SELECT designation',
            'specified in any DVQSL level'
        call FROM_ERROR
      end
  /*-- must have at least one file specified by the FROM/input statement --*/
  if JF = 0    then
      do
        say '  Must have at least one FROM input file',
            'specified in any DVQSL level'
        call FROM_ERROR
      end
  return

/*-------------------------------------------------------------------------*\
|*       - Compiler verb processing of the inputted SQL commands -          |
|*  The following paragraphs are for the parsing of the input commands      |
|*  into the tokens used by the SQL processor to do its thing.              |
|*  Each section deals with one of the main verbs, SELECT, FROM, INTO,      |
```

```
/*  WHERE, or ORDER BY.                                                    */
\*-------------------------------------------------------------------------*/
SELECT_VERB:
  SEL_OPTS = ''
  SEL_FLDS = ''
  do forever
    if substr(PARS_LINE,1,1) = "'"    then  /* literal value processing */
        do
          AA = pos("'",PARS_LINE,2)
          if AA > 0    then
              do
                JS = JS + 1
                interpret "SELECT#"I".JS = substr(PARS_LINE,1,AA)"
                if substr(PARS_LINE,AA+1,1) = ','    then
                    do
                      PARS_LINE = substr(PARS_LINE,AA+2) /* drop comma and any*/
                      PARS_LINE = strip(PARS_LINE,'l')   /* following blanks */
                      iterate
                    end
                else
                    do
                      PARS_LINE = substr(PARS_LINE,AA+1)
                      return
                    end
              end
          else
              do
                say '  unbalanced quotes bounding a literal value'
                call SELECT_ERROR
              end
        end
    parse upper var PARS_LINE FIELD_DEF PARS_LINE
    select
      when FIELD_DEF = ''    then
          do
            call READ_SYSIN
            iterate
          end
      when FIELD_DEF = '*'    then    /* select entire record */
          do
            JS = JS + 1
            interpret "SELECT#"I".JS = '*'"
          end
      when FIELD_DEF = 'DISTINCT'    then
        do
          JS = JS + 1
          interpret "SELECT#"I".JS = 'DISTINCT'"
          iterate
        end
      when substr(FIELD_DEF,1,6) = 'COUNT('    then
        do
          if I > 1    then
              if SEL_OPTS = 'Y'    then
                  do
                    say '  SELECT Options (MIN, MAX, COUNT, AVG, SUM) are',
                        'mutually exclusive at the subquery level'
                    call SELECT_ERROR
                  end
          SEL_OPTS = 'Y'
          if SEL_FLDS = 'Y'    then
              do
                say '  SELECT Options (MIN, MAX, and COUNT) are mutually',
                    'exclusive with subset selection'
                call SELECT_ERROR
              end
          JS = JS + 1
          interpret "SELECT#"I".JS = 'COUNT'"
          FIELD_DEF = substr(FIELD_DEF,7)
          if FIELD_DEF = 'DISTINCT'    then
              do
                JS = JS + 1
                interpret "SELECT#"I".JS = 'DISTINCT'"
                parse upper var PARS_LINE FIELD_DEF PARS_LINE
              end
          if substr(FIELD_DEF,1,1) = '*'    then
              do
                JS = JS + 1
                interpret "SELECT#"I".JS = '*'"
```

```
          PARS_LINE = substr(FIELD_DEF,2)||PARS_LINE /* re-attach any */
        end                                          /* trailing characters */
      else
        do
          call FIELD_FORMAT_CHECK
          if FIELD_ERR = 'Y'    then
              call SELECT_ERROR
          JS = JS + 1
          interpret "SELECT#"I".JS = AA'.,'BB','CC"
        end
      if substr(PARS_LINE,1,1) = ')'    then
        PARS_LINE = substr(PARS_LINE,2)
      else
        do
          say '  COUNT format error - missing closing paren'
          call SELECT_ERROR
        end
    end
  when substr(FIELD_DEF,1,4) = 'MAX('    then
    do
      if I > 1    then
        if SEL_OPTS = 'Y'    then
            do
              say '  SELECT Options (MIN, MAX, COUNT, AVG, SUM) are',
                  'mutually exclusive at the subquery level'
              call SELECT_ERROR
            end
      SEL_OPTS = 'Y'
      if SEL_FLDS = 'Y'    then
        do
          say '  SELECT Options (MIN, MAX, and COUNT) are mutually',
              'exclusive with subset selection'
          call SELECT_ERROR
        end
      JS = JS + 1
      interpret "SELECT#"I".JS = 'MAX'"
      FIELD_DEF = substr(FIELD_DEF,5)
      call FIELD_FORMAT_CHECK
      if FIELD_ERR = 'Y'    then
        call SELECT_ERROR
      JS = JS + 1
      interpret "SELECT#"I".JS = AA'.,'BB','CC"
      if substr(PARS_LINE,1,1) = ')'    then
        PARS_LINE = substr(PARS_LINE,2)
      else
        do
          say '  MAX format error - missing closing paren'
          call SELECT_ERROR
        end
    end
  when substr(FIELD_DEF,1,4) = 'MIN('    then
    do
      if I > 1    then
        if SEL_OPTS = 'Y'    then
            do
              say '  SELECT Options (MIN, MAX, COUNT, AVG, SUM) are',
                  'mutually exclusive at the subquery level'
              call SELECT_ERROR
            end
      SEL_OPTS = 'Y'
      if SEL_FLDS = 'Y'    then
        do
          say '  SELECT Options (MIN, MAX, and COUNT) are mutually',
              'exclusive with subset selection'
          call SELECT_ERROR
        end
      JS = JS + 1
      interpret "SELECT#"I".JS = 'MIN'"
      FIELD_DEF = substr(FIELD_DEF,5)
      call FIELD_FORMAT_CHECK
      if FIELD_ERR = 'Y'    then
        call SELECT_ERROR
      JS = JS + 1
      interpret "SELECT#"I".JS = AA'.,'BB','CC"
      if substr(PARS_LINE,1,1) = ')'    then
        PARS_LINE = substr(PARS_LINE,2)
      else
        do
```

```
          say '  MIN format error - missing closing paren'
          call SELECT_ERROR
        end
    end
  when substr(FIELD_DEF,1,4) = 'AVG('    then    /* not functional */
    do
      if I > 1    then
          if SEL_OPTS = 'Y'    then
              do
                say '  SELECT Options (MIN, MAX, COUNT, AVG, SUM) are',
                    'mutually exclusive at the subquery level'
                call SELECT_ERROR
              end
      SEL_OPTS = 'Y'
      if SEL_FLDS = 'Y'    then
          do
            say '  SELECT Options (MIN, MAX, and COUNT) are mutually',
                'exclusive with subset selection'
            call SELECT_ERROR
          end
      JS = JS + 1
      interpret "SELECT#"I".JS = 'AVG'"
      FIELD_DEF = substr(FIELD_DEF,5)
      call FIELD_FORMAT_CHECK
      if FIELD_ERR = 'Y'    then
          call SELECT_ERROR
      JS = JS + 1
      interpret "SELECT#"I".JS = AA'.,'BB','CC"
      if substr(PARS_LINE,1,1) = ')'    then
          PARS_LINE = substr(PARS_LINE,2)
      else
          do
            say '  AVG format error - missing closing paren'
            call SELECT_ERROR
          end
    end
  when substr(FIELD_DEF,1,4) = 'SUM('    then    /* not functional */
    do
      if I > 1    then
          if SEL_OPTS = 'Y'    then
              do
                say '  SELECT Options (MIN, MAX, COUNT, AVG, SUM) are',
                    'mutually exclusive at the subquery level'
                call SELECT_ERROR
              end
      SEL_OPTS = 'Y'
      if SEL_FLDS = 'Y'    then
          do
            say '  SELECT Options (MIN, MAX, and COUNT) are mutually',
                'exclusive with subset selection'
            call SELECT_ERROR
          end
      JS = JS + 1
      interpret "SELECT#"I".JS = 'SUM'"
      FIELD_DEF = substr(FIELD_DEF,5)
      call FIELD_FORMAT_CHECK
      if FIELD_ERR = 'Y'    then
          call SELECT_ERROR
      JS = JS + 1
      interpret "SELECT#"I".JS = AA'.,'BB','CC"
      if substr(PARS_LINE,1,1) = ')'    then
          PARS_LINE = substr(PARS_LINE,2)
      else
          do
            say '  SUM format error - missing closing paren'
            call SELECT_ERROR
          end
    end
  otherwise
    do
      SEL_FLDS = 'Y'
      if SEL_OPTS = 'Y'    then
          do
            say '  SELECT Options (MIN, MAX, and COUNT) are mutually',
                'exclusive with subset selection'
            call SELECT_ERROR
          end
      call FIELD_FORMAT_CHECK
```

```
        if FIELD_ERR = 'Y'    then
            call SELECT_ERROR
        JS = JS + 1
        interpret "SELECT#"I".JS = AA'.,'BB','CC"
      end
    end
    if substr(PARS_LINE,1,1) = ','    then    /* more fields to come */
        PARS_LINE = substr(PARS_LINE,2)
    else                                  /* no continuation, must be done */
        return
  end
  return

FIELD_FORMAT_CHECK:
  FIELD_ERR = ''
  parse var FIELD_DEF AA '.(' BB ',' CC ')' DD
  if AA = '*' | AA = '(*)'    then return    /* select the entire record */
  select
    when datatype(substr(AA,1,1),'U') = 0    then
      do
        say '  file alias does not start with an ALPHABETIC character'
        FIELD_ERR = 'Y'
      end
    when datatype(AA,'A') = 0    then
      do
        say '  file alias is not all ALPHA-NUMERIC characters'
        FIELD_ERR = 'Y'
      end
    when length(AA) > 4    then
      do
        say '  file alias is more than 4 characters'
        FIELD_ERR = 'Y'
      end
    otherwise nop
  end
  if datatype(BB,'W') = 0    then
      do
        say '  start column/displacement is not numeric'
        FIELD_ERR = 'Y'
      end
  if datatype(CC,'W') = 0    then
      do
        say '  length is not numeric'
        FIELD_ERR = 'Y'
      end
  PARS_LINE = DD||PARS_LINE  /* re-attach any trailing characters such as a */
                             /* comma, right-paren, or next field           */
  return

SELECT_ERROR:
  say ' FIELD DEFINITION ERROR - 'FIELD_DEF
  say ' '
  call SELECT_FORMAT
  exit 8

SELECT_FORMAT:
  say ' formats: SELECT COUNT({DISTINCT} field), MIN(field), MAX(field)'
  say '          SELECT {DISTINCT} field1, field2, etc, fieldx'
  say ' field: A.(displacement,length),   where...'
  say '      A  - DDname alias (1 to 4 ALPHA-NUMERIC chars)'
  say '      displacement  - location of the field in the record'
  say '      length  - length of field at specified location"
  say ' '
  return

FROM_VERB:
  do forever
    CPOS = pos(',',PARS_LINE)         /* check for multi-file delimiter */
    if CPOS = 0    then               /* if none, */
        CPOS = pos(')',PARS_LINE)     /* check for end-of-level delimiter */
    if CPOS > 0    then               /* if delimiter found, parse accordingly */
        do
          DDNAME_ALIAS = strip(substr(PARS_LINE,1,CPOS-1),'b')
          PARS_LINE = substr(PARS_LINE,CPOS)
          parse var DDNAME_ALIAS DDNAME DDALIAS
        end
    else
        parse upper var PARS_LINE DDNAME DDALIAS PARS_LINE
```

```
if DDNAME = ''    then
    do
      call READ_SYSIN
      iterate
    end
if length(DDNAME) > 8    then
    do
      say '  Invalid DDname  -  DDname more than eight characters'
      call FROM_ERROR
    end
if datatype(DDNAME,'A') = 0  |  datatype(substr(DDNAME,1,1),'U') = 0  then
    do
      say '  Invalid DDname  -  Must be 1 to 8 alphanumeric characters',
          'with first character being alphabetic'
      call FROM_ERROR
    end
if length(DDALIAS) = 0    then
    do
      say '  no file alias was specified - it is mandatory in DVSQL'
      call FROM_ERROR
    end
if datatype(substr(DDALIAS,1,1),'U') = 0    then
    do
      say '  file alias does not start with an ALPHABETIC character'
      call FROM_ERROR
    end
if datatype(DDALIAS,'A') = 0    then
    do
      say '  file alias is not all ALPHA-NUMERIC characters'
      call FROM_ERROR
    end
/*-- check for duplicate DDNAME or DDALIAS for the current level --*/
if JF > 0    then
    do II = 1 to JF
      interpret "DDNAME_DDALIAS = FROM#"I".II"
      parse var DDNAME_DDALIAS XXXX YYYY
      if XXXX = DDNAME    then
          do
            say '  Duplicate DDname 'DDNAME' encountered for this level'
            call FROM_ERROR
          end
      if YYYY = DDALIAS    then
          do
            say '  The DD alias 'DDALIAS' was already assigned to 'XXXX
            call FROM_ERROR
          end
    end
JF = JF + 1
interpret "FROM#"I".JF = DDNAME' 'DDALIAS" /* add to this level's list */
/*-- check for full request DDname/alias confilcts --*/
/*-- All levels of FROM information, up to this point, are scanned for --*/
/*-- conflicts in prior DDname/alias information... precisely, the same --*/
/*-- DDname with more than one alias or an alias wih more than one DDname */
if DDS = 0    then
    II = 1
else
    do
      do II = 1 to DDS
        if INALIAS.II = DDALIAS    then
            if INDD.II = DDNAME    then
                leave
            else
                do
                  say '  An alias may reference only one DDname'
                  say '  The alias 'INALIAS.II' has already been assigned',
                      'to 'INDD.II
                  call FROM_ERROR
                end
      end
      do II = 1 to DDS
        if INDD.II = DDNAME    then
            if INALIAS.II = DDALIAS    then
                leave
            else
                do
                  say '  Only one alias is allowed per DDname'
                  say '  'DDNAME' already has the',
                      'alias 'INALIAS.II' assigned'
```

```
                    call FROM_ERROR
                  end
          end
        end
    if II > DDS    then
        do
          DDS = II
          INDD.0 = II
          INALIAS.0 = II
          INDD.II = DDNAME
          INALIAS.II = DDALIAS
        end
    if substr(PARS_LINE,1,1) = ','    then   /* if multi-file delim found, */
        PARS_LINE = substr(PARS_LINE,2)      /* drop delim and continue    */
    else
        do
          /*-- validate SELECT field file references --*/
          do II = 1 to JS
            interpret "FIELD_DEF = SELECT#"I".II"
            if substr(FIELD_DEF,1,1) \= "'"    then
                if pos(',',FIELD_DEF) > 0    then
                    do
                      parse var FIELD_DEF AA '.,' BB ',' CC
                      do JJ = 1 to JF
                        interpret "DDNAME_DDALIAS = FROM#"I".JJ"
                        parse var DDNAME_DDALIAS DDNAME DDALIAS
                        if DDALIAS = AA    then
                            leave
                      end
                      if JJ > JF    then
                          do
                            FIELD_DEF = AA'.('BB','CC')'
                            say 'The alias used in 'FIELD_DEF' is not one',
                                'specified in the current level FROM statement'
                            call SELECT_ERROR
                          end
                    end
          end
          return                                    /* done with FROM */
        end
  end
  return

FROM_ERROR:
  say ' INPUT FILE DESIGNATION ERROR - 'DDNAME' 'DDALIAS
  say '     invalid file designation format under a FROM verb'
  say ' '
  call FROM_FORMAT
  exit 8

FROM_FORMAT:
  say ' format: FROM DDname1 A1, DDname2 A2, etc.'
  say '     DDname     - DDname of the file to use as input'
  say '     Ax         - alias assigned by the user to the DDname. It must be'
  say '                  an alpha-numeric name that starts with an alphabetic'
  say '                  character.  No limit on length.'
  say ' '
  return

INTO_VERB:
  if I > 1    then
      do
        say '**ERROR**  Use of the INTO verb is not valid for sub-queries.'
        say '    It is valid on the primary SQL level ONLY'
        exit 8
      end
  parse upper var PARS_LINE DDNAME PARS_LINE
  if length(DDNAME) > 8    then
      do
        say '  Invalid DDname  -  DDname more than eight characters'
        call INTO_ERROR
      end
  if datatype(DDNAME,'A') = 0  |  datatype(substr(DDNAME,1,1),'U') = 0  then
      do
        say '  Invalid DDname  -  Must be 1 to 8 alphanumeric characters',
            'with first character being alphabetic'
        call INTO_ERROR
      end
```

```
  OUT#DD = DDNAME
  return

INTO_ERROR:
  say ' OUTPUT FILE DESIGNATION ERROR - 'DDNAME
  say '     invalid file designation format under a INTO verb'
  say ' '
  call INTO_FORMAT
  exit 8

INTO_FORMAT:
  say ' format: INTO DDname'
  say '     DDname   - DD name of the file to use as output'
  say ' '
  return

WHERE_VERB:
  do forever
    /*---- check for literal values ----*/
    select
      when substr(PARS_LINE,1,1) = "'"    then
          do
            CPOS = pos("'",PARS_LINE,2)
            if CPOS > 0    then
                do
                  JW = JW + 1
                  interpret "WHERE#"I".JW = substr(PARS_LINE,1,CPOS)"
                  PARS_LINE = substr(PARS_LINE,CPOS+1)
                  iterate
                end
            else
                do
                  say '  unbalanced quotes bounding a literal value'
                  call WHERE_ERROR
                end
          end
    /*---- check for level changes and such ----*/
      when substr(PARS_LINE,1,1) = "("    then
          if substr(PARS_LINE,2,1) = "'"   then  /* user table list for IN */
              do
                call DEFINE_USER_TABLE  /* parse off table values */
                iterate
              end
          else
              do        /* embedded SQL identified */
                JW = JW + 1
                interpret "WHERE#"I".JW = 'SUBQRY#"IMAX+1"'"
                return    /* return to main loop to go up a DVSQL level */
              end
      when substr(PARS_LINE,1,1) = ")"    then  /* end of SQL level */
          return          /* return to main loop to go back a DVSQL level */
      otherwise nop
    end

    parse upper var PARS_LINE XXXX PARS_LINE /* get next data group */
    if XXXX = ''    then    /* if end of line, get next input record */
        do
          call READ_SYSIN
          iterate
        end
    /*---- check for valid operand data values ----*/
    select
      when XXXX = 'IN'    then
          do
            JW = JW + 1
            interpret "WHERE#"I".JW = 'IN'"
          end
      when XXXX = 'NOT'    then
          do
            if substr(PARS_LINE,1,3) = 'IN '    then
                do
                  parse upper var PARS_LINE XXXX PARS_LINE
                  JW = JW + 1
                  interpret "WHERE#"I".JW = 'NI'"
                end
            else
                do
                  say '  expecting NOT IN... found NOT ????'
```

```
            call WHERE_ERROR
          end
    end
when XXXX = 'AND'    then
    do
      JW = JW + 1
      interpret "WHERE#"I".JW = 'AND'"
    end
when XXXX = 'OR'    then
    do
      JW = JW + 1
      interpret "WHERE#"I".JW = 'OR'"
    end
when XXXX = '='    then
    do
      JW = JW + 1
      interpret "WHERE#"I".JW = 'EQ'"
    end
when XXXX = '<>'   then
    do
      JW = JW + 1
      interpret "WHERE#"I".JW = 'NE'"
    end
when XXXX = '<'    then
    do
      JW = JW + 1
      interpret "WHERE#"I".JW = 'LT'"
    end
when XXXX = '<='   then
    do
      JW = JW + 1
      interpret "WHERE#"I".JW = 'LE'"
    end
when XXXX = '>'    then
    do
      JW = JW + 1
      interpret "WHERE"I".JW = 'GT'"
    end
when XXXX = '>='   then
    do
      JW = JW + 1
      interpret "WHERE"I".JW = 'GE'"
    end
when XXXX = 'ALPHA'  |  XXXX = 'ALPHABETIC'    then
    do
      JW = JW + 1
      interpret "WHERE#"I".JW = '#ALPHA'"
    end
when XXXX = 'ALPHANUMERIC'    then
    do
      JW = JW + 1
      interpret "WHERE#"I".JW = '#ALPHANUMERIC'"
    end
when XXXX = 'BETWEEN'    then
    do
      JW = JW + 1
      interpret "WHERE#"I".JW = '#BETWEEN'"
    end
when XXXX = 'INTEGER'   then
    do
      JW = JW + 1
      interpret "WHERE#"I".JW = '#INTEGER'"
    end
when XXXX = 'ORDER'    then
    do
      PARS_LINE = 'ORDER 'PARS_LINE
      return
    end
otherwise
    do
      FIELD_DEF = XXXX
      call FIELD_FORMAT_CHECK
      if FIELD_ERR = 'Y'    then
          call WHERE_ERROR
      /*-- validate file alias reference --*/
      do JJ = 1 to DDS
        if AA = INALIAS.JJ    then
            leave
```

```
            end
            if JJ > DDS    then
                do
                  say 'The file alias used in 'FIELD_DEF' is not one specfied',
                      'in current or prior level FROM statements'
                  call SELECT_ERROR
                end
            /*-- add the field to the WHERE list --*/
            JW = JW + 1
            interpret "WHERE#"I".JW = AA'.','BB','CC"
          end
      end
    end
    return

WHERE_ERROR:
    say ' WHERE - EQUATION ERROR - 'XXXX
    say '     invalid equation part under a WHERE verb'
    say ' '
    call WHERE_FORMAT
    exit 8

WHERE_FORMAT:
    say ' format: WHERE cond1 {AND/OR cond2}'
    say '     condx  =  field1 operator field2'
    say '               field1 operator (sub-query)'
    say '               field1 class-test'
    say '               field1 IN/NOT IN user-table'
    say '               field1 IN/NOT IN (sub-query)'
    say "               field1 BETWEEEN 'value1' AND 'value2'"
    say '     field1 format: Ax.(displacement,length)          where...'
    say '         Ax  -  alias reference to input file'
    say '         displacement  -  location of the field in the record'
    say '         length  -  length of field at the specified location in the ',
        'record'
    say '     operator: =, <>, <, <=, >, >='
    say "     field2 format: Ax.(displacement,length)  or 'literal-value'"
    say '     class-test: ALPHABETIC/ALPHA all upper alphabetic characters'
    say '                 ALPHANUMERIC all whole number or upper alphabetic',
        'characters'
    say '                 INTEGER all whole numbers'
    say "     user-table format: ('lit1','lit2','lit3',...,'litx')"
    say '         litx is a literal/character value'
    say '     sub-query format: another query that generates a single value'
    say '         or table/set of values to be used for comparison'
    say ' '
    return

ORDER_BY_VERB:
/*-------------------------------------------------------------------------*\
|  parse information from the ORDER BY control card/s and convert into the  |
|  SYNCSORT format needed to for processing                                 |
\*-------------------------------------------------------------------------*/
    if I > 1    then
        do
          say '**ERROR**  Use of the ORDER BY verb is not valid for sub-queries.'
          say '    It is valid on the primary SQL level ONLY'
          exit 8
        end
    SORT#CARD = '  SORT FIELDS=('
    parse upper var PARS_LINE AA PARS_LINE
    if AA ¬= 'BY'    then call ORDER_BY_ERROR
    do forever
      if PARS_LINE = ''    then
          do
            call READ_SYSIN
            iterate
          end
      parse upper var PARS_LINE '(' BB ',' CC ')' PARS_LINE
      if datatype(BB,'W') = 0    then
          do
            say '  start column is not numeric'
            call ORDER_BY_ERROR
          end
      if datatype(CC,'W') = 0    then
          do
            say '  end column is not numeric'
            call ORDER_BY_ERROR
```

```
        end
    PARS_LINE = strip(PARS_LINE,'l')
    select    /* set ordering default if needed */
      when PARS_LINE = ''    then
          DD = 'A'
      when substr(PARS_LINE,1,1) = ','    then
          do
            DD = 'A,'
            PARS_LINE = substr(PARS_LINE,2)
          end
      when substr(PARS_LINE,1,3) = 'ASC'   then
          if substr(PARS_LINE,4,1) = ','    then
              do
                DD = 'A,'
                PARS_LINE = substr(PARS_LINE,5)
              end
          else
              do
                DD = 'A'
                PARS_LINE = substr(PARS_LINE,4)
              end
      when substr(PARS_LINE,1,4) = 'DESC'    then
          if substr(PARS_LINE,5,1) = ','    then
              do
                DD = 'D,'
                PARS_LINE = substr(PARS_LINE,6)
              end
          else
              do
                DD = 'D'
                PARS_LINE = substr(PARS_LINE,5)
              end
      otherwise
          do
            say '  field sort order is not ASCending or DESCending'
            call ORDER_BY_ERROR
          end
    end
    if length(DD) = 1    then
        do
          SORT#CARD = SORT#CARD||BB','CC',CH,'DD')'
          return
        end
    else
        SORT#CARD = SORT#CARD||BB','CC',CH,'DD
  end
  return

ORDER_BY_ERROR:
  say ' ORDER BY - FIELD DEFINITION ERROR - 'FIELD_DEF
  say '     invalid field definition under a ORDER BY verb'
  say ' '
  call ORDER_BY_FORMAT
  exit 8

ORDER_BY_FORMAT:
  say '     format: ORDER BY field1, field2, field3, etc.    where...'
  say '         fieldx is (displacement,length){ order}
  say '         displacement - location of field in the output record'
  say '         length       - length of field at specified location'
  say '         order        - order to sort specified field
  say '                ASC (ASCending) - default'
  say '                DESC (DESCending)'
  say ' '
  return

DEFINE_USER_TABLE:
/*-------------------------------------------------------------------------*\
|    Parse off the user supplied values of a WHERE IN option and put them    |
|    into the processing stack under a TABLE type.                           |
\*-------------------------------------------------------------------------*/
  UTBL_CNT = UTBL_CNT + 1        /* increase the stored user table counter */
  BKT_CNT = 0                       /* reset table bucket count to 0 */
  PARS_LINE = substr(PARS_LINE,3)    /* point after the 1st quote */
  do forever
    QPOS = pos(QUOT,PARS_LINE)
    if QPOS > 0    then
        do
```

```
        AA = substr(PARS_LINE,1,QPOS-1)
        PARS_LINE = substr(PARS_LINE,QPOS+1)
      end
    else
        AA = PARS_LINE
    BKT_CNT = BKT_CNT + 1                  /* add a bucket to the table */
    interpret "UTBL@"UTBL_CNT"."BKT_CNT" = '"AA"'" /*put data into the bucket*/
    select
      when substr(PARS_LINE,1,1) = ','     then  /* if another value follows, */
        if substr(PARS_LINE,2,1) = "'"     then  /* immediately,              */
            PARS_LINE = substr(PARS_LINE,3)     /* strip off leading quote */
        else
            do
              call READ_SYSIN           /* otherwise, get next input card */
              if substr(PARS_LINE,1,1) = "'"    then
                  PARS_LINE = substr(PARS_LINE,2) /* and strip off leading ' */
            end
      when substr(PARS_LINE,1,1) = ')'     then
          do
            PARS_LINE = substr(PARS_LINE,2)     /* strip off leading ) */
            leave                               /* - done building table - */
          end
      otherwise                                 /* ERROR ERROR ERROR */
        do
          say ' WHERE a.(displ,length) IN list ERROR'
          say '      invalid format of user provided value list'
          say "      format of provided list: ('value','value',etc. ,'value')"
          say '        Continuation of values may span multiple lines as long'
          say '        as each line ends with a comma to indicate more values'
          say '        are provided on following lines.'
          exit 8
        end
    end
  end
  interpret "UTBL@"UTBL_CNT".0 = "BKT_CNT          /* set table index counter */
  JW = JW + 1
  interpret "WHERE#"I".JW = 'UTBL@"UTBL_CNT"'"     /*add table item to stack*/
  return

PROCESS_SQL_LEVEL:
/*-------------------------------------------------------------------------*\
|    Look at the options given from the current stack and do necessary      |
|    processing to accomplish the task.                                     |
|---------------------------------------------------------------------------|
|  WHERE can have one of the following formats:                             |
|    A. #            do a data class test                                   |
|    A. operator *   look for file A records with a given literal value     |
|    A. operator A.  look for file A record with the same value in two places|
|    A. operator B.  look for records in file B that have one of the values |
|                    in the designated master list file A                   |
\*-------------------------------------------------------------------------*/
do XI = 1 to JF
  interpret "DDNAME_DDALIAS = FROM#"I".XI"
  say 'FROM#'I'.'XI' = 'DDNAME_DDALIAS
end

say ' WHERE_DATA = 'WHERE_DATA
say ' DISTINCT  = 'DISTINCT
say ' OUT#FLD.0 = 'OUT#FLD.0
say ' SORT#CARD = 'SORT#CARD
say ' IN_FILE.0 = 'IN_FILE.0

  MERG_LIST.0 = 0
  AND_FILE = ''

  do forever  /*-- main loop of processing WHERE_DATA --*/
    /*-- check for setting "reuse" file for AND processing --*/
    if substr(WHERE_DATA,1,4) = 'AND'    then
        do
          AND_FILE = WORKFILE                   /* set re-use file */
          parse var WHERE_DATA JUNK WHERE_DATA  /* strip off the AND */
        end
    else
        do
          AND_FILE = ''
          /*-- check to put workfile on MERG_LIST stack for OR processing --*/
          if substr(WHERE_DATA,1,3) = 'OR '    then
              do
```

```
            MERG_LIST.0 = MERG_LIST.0 + 1                  /* add file to */
            interpret "MERG_LIST."MERG_LIST.0" = WORKFILE" /* merge stack */
            parse var WHERE_DATA JUNK WHERE_DATA    /* strip off the OR */
          end
      end

   parse var WHERE_DATA FLD1 OPER WHERE_DATA /* obtain field1 and operator */
say '++++++++ FLD1='FLD1'    OPER='OPER'    WHERE_DATA='WHERE_DATA

   /*-- attempt to preset FLD2 if needed --*/
   select
     when FLD1 = ''    then nop            /* select reformat only */
     when substr(OPER,1,1) = '#'    then   /* if indicates class test, done */
         nop
     when substr(WHERE_DATA,1,1) = "'"    then    /* literal */
        do
          CPOS = pos("'",WHERE_DATA,2) /* extract literal value... */
          FLD2 = substr(WHERE_DATA,1,CPOS)  /* could have embedded blanks */
          WHERE_DATA = substr(WHERE_DATA,CPOS+1)
        end
     otherwise    /* user table, subquery result, or field specification */
        do
          parse var WHERE_DATA FLD2 WHERE_DATA
          if substr(FLD2,1,7) = 'SUBQRY#'    then   /* if subquery result, */
              /*-- replace with useable literal or field value --*/
              do                              /* determine variable value. */
                SQ_NUM = substr(FLD2,8) /* determine originating query nmbr */
                interpret "FLD2 = "FLD2            /* set the stored value */
                if substr(FLD2,1,1) = '¢'    then /* if is function result, */
                    FLD2 = "'"substr(FLD2,2)"'" /* reset FLD2 as a literal. */
                else
                    do                      /* otherwise, obtain file info */
                      x = listdsi("'"FLD2"'")    /* and set file alias and */
                    interpret "FLD2 = 'SQ#"SQ_NUM".,1,"SYSLRECL"' " /*field*/
                    end
              end
        end
   end

say '++ AND_FILE = 'AND_FILE
say '++ COMPOUND = 'COMPOUND
say '++++++++ FLD1='FLD1'    OPER='OPER'    FLD2='FLD2
   select
     when FLD1 = ''    then
       leave
     when substr(OPER,1,1) = '#'    then
       do    /* process a class test selection */
say '++++++++ CLASS TEST ++++++++'
         parse var FLD1 DDALIAS '.,' AA ',' BB
         if COMPOUND = 'Y'   then           /* if using compound file */
             if DDALIAS = DDALIAS.2  then   /* if field in 2nd part of file */
                 AA = AA + DDLRECL.1        /* adjust the displacement */
         drop SORTCARD.
         SORTCARD.0 = 0
         call CLASS_COMPARE         /* gen necessary INCLUDE cards */
         SORTCARD.0 = SORTCARD.0 + 1
         interpret "SORTCARD."SORTCARD.0" = '  SORT FIELDS=COPY '"
         /*---- select records/data by data class ----*/
         if AND_FILE = ''    then
             call SORT_DATA 'SETUP' DSN.DDALIAS
         else
             call SORT_DATA 'SETUP' AND_FILE
         WORKFILE = SORTOUT
       end
     when substr(FLD2,1,1) = "'"    then
       do    /* process a sort select on a literal value */
say '++++++++ LITERAL COMPARE ++++++++'
         parse var FLD1 DDALIAS '.,' AA ',' BB
         if COMPOUND = 'Y'   then           /* if using compound file */
             if DDALIAS = DDALIAS.2  then   /* if field in 2nd part of file */
                 AA = AA + DDLRECL.1        /* adjust the displacement */
         drop SORTCARD.
         SORTCARD.0 = 2
         SORTCARD.1 = "  INCLUDE COND=("AA","BB",CH,"OPER",C"FLD2")"
         SORTCARD.2 = '  SORT FIELDS=COPY '
         /* select records from the file */
         if AND_FILE = ''    then
             call SORT_DATA 'SETUP' DSN.DDALIAS
```

```
          else
              call SORT_DATA 'SETUP' AND_FILE
          WORKFILE = SORTOUT
        end
      when substr(FLD2,1,5) = 'UTBL@'    then
        do    /* compare a file to a user specified list of values */
say '++++++++ USER TABLE COMPARE ++++++++'
          parse var FLD1 DDALIAS ',' AA ',' BB
          if COMPOUND = 'Y'   then           /* if using compound file */
              if DDALIAS = DDALIAS.2  then   /* if field in 2nd part of file */
                  AA = AA + DDLRECL.1        /* adjust the displacement */
          UTABL_NUM = substr(FLD2,5)  /* determine which user table */
          drop SORTCARD.
          SORTCARD.0 = 0
          /*---- build the INCLUDE/OMIT sort cards ----*/
          interpret "T1 = UTBL@"UTABL_NUM".0"
          if OPER = 'IN'    then
              IOTYP = 'INCLUDE COND=('
          else
              IOTYP = 'OMIT COND=('
          do T2 = 1 to T1
            if T2 = 1    then
                TMPCARD = '  'IOTYP
            else
                TMPCARD = '            '
            interpret "T3 = UTBL@"UTABL_NUM"."T2
            TMPCARD = TMPCARD||AA","BB",CH,EQ,C'"T3"'"
            if T2 = T1    then
                TMPCARD = TMPCARD')'
            else
                TMPCARD = TMPCARD',OR,'
            interpret 'SORTCARD.'T2' = TMPCARD'
          end
          SORTCARD.0 = T1
          SORTCARD.0 = SORTCARD.0 + 1
          interpret "SORTCARD."SORTCARD.0" = '  SORT FIELDS=COPY '"
          /* select/exclude records from the file */
          if AND_FILE = ''    then
              call SORT_DATA 'SETUP' DSN.DDALIAS
          else
              call SORT_DATA 'SETUP' AND_FILE
          WORKFILE = SORTOUT
        end
      otherwise
        do
          parse var FLD1 DDALIAS1 '.,' AA ',' BB
          parse var FLD2 DDALIAS2 '.,' CC ',' DD
          if DDALIAS1 = DDALIAS2    then
              call CMPR_2_FIELDS_SAME_FILE
          else
              if COMPOUND = 'Y'    then
                  if DDALIAS1 = DDALIAS.1  |  DDALIAS1 = DDALIAS.2    then
                      if DDALIAS2 = DDALIAS.1  |  DDALIAS2 = DDALIAS.2    then
                          call CMPR_2_FIELDS_SAME_FILE
                      else
                          call CMPR_2_FIELDS_DIFF_FILES
                  else
                      call CMPR_2_FIELDS_DIFF_FILES
              else
                  call CMPR_2_FIELDS_DIFF_FILES
        end
    end
  end

/*---- processing for merging of ORed outputs ----*/
  if MERG_LIST.0 > 0    then
      do
        MERG_LIST.0 = MERG_LIST.0 + 1        /* add the last workfile output */
        interpret "MERG_LIST."MERG_LIST.0" = WORKFILE" /* to the merge stack */
        say ' '
        say ' -- Merging of ORed comparison outputs --'
        drop SORTCARD.
        SORTCARD.1 = '  SORT FIELDS=COPY '
        SORTCARD.0 = 1
        /*---- merge OR comparison outputs ----*/
        call SORT_DATA 'SETUP' '*MERGE*'  /* tell SORT to use MERGE */
        WORKFILE = SORTOUT                  /* stack for inputs */
      End
```

SELECTING DATA - c)

The following twelve statements select data from the selected combined like-keyed records. This is accomplished by converting the user specified SELECT criteria into SYNCSORT "INREC" control cards via paragraph GEN_INREC_CARD (see Selecting Data c 2 for more detailed information), checking for the user specified requirement of "DISTINCT" output records in which case a SYNCSORT "SUM FIELDS=NONE" control card will be added via paragraph DISTINCT_CHECK, and processing the generated SYNCSORT control cards against the temporary file of selected combined records to parse out the specified SELECT data via paragraph SORT_DATA. The resulting information is again stored on another temporary work file.

```
  /*-- SELECT reformatting and functions... etc. --*/
  if OUT#FLD.1 = '*'  &  DISTINCT = ''    then   /* no use in just recopying */
      nop                                        /* the file to another      */
  else
      do
        drop SORTCARD.
        SORTCARD.0 = 0
        call GEN_INREC_CARD              /* check for reformatting needs */
        call DISTINCT_CHECK WORKFILE     /* check for duplicate elimination */
```

TEMPORARY OUTPUTTING - d)

The following statement processes the data SELECTion criteria against the temporary selected combined-records file and writes the resulting output to another temporary output file/data set. See Temporary_Outputting_SORT_DATA for more detail.

```
        call SORT_DATA 'PROCESS' WORKFILE
        WORKFILE = SORTOUT
      end

  /*-- SELOPT#x is used by higher level SQLs to access to result of a lower */
  /*-- level MAX, MIN, COUNT, etc. SELECT option ---------------------------*/
  if SEL_OPT = ''    then
     interpret "SELOPT#"I" = ''"  /* set this level's SEL_OPT output to null */
  else
      /*-- process this level's SELECT option and store output in SELOPT#x */
      do
        call PROCESS_A_FUNCTION SEL_OPT
        interpret "SELOPT#"I" = result"
      end

  return

CMPR_2_FIELDS_SAME_FILE:
/*-------------------------------------------------------------------------*\
|  This routine processes a single file that compares two fields within the  |
|  same file... this could be a reference to two fields within the same file |
|  or a reference to two fields in separate files that are both part of a    |
|  compound record in which the data from both files exists.  Either way,    |
|  the displacements to the fields are adjusted as necessary and the file    |
|  (singular or compound) is processed thru a single pass of SORT to         |
|  accomodate the fields comparison.                                         |/
\*-------------------------------------------------------------------------*/
say '++++++++ COMPARE 2 FIELDS IN SAME FILE ++++++++'
  if COMPOUND = 'Y'    then
      do
        if DDALIAS1 = DDALIAS.2    then
            AA = AA + DDLRECL.1
        if DDALIAS2 = DDALIAS.2    then
            CC = CC + DDLRECL.1
      end
  drop SORTCARD.
  SORTCARD.0 = 2
  SORTCARD.1 = '  INCLUDE COND=('AA','BB',CH,'OPER','CC','DD',CH)'
  SORTCARD.2 = '  SORT FIELDS=COPY '
  /* select records from the file */
  if AND_FILE = ''    then
      call SORT_DATA 'SETUP' DSN.DDALIAS
  else
      call SORT_DATA 'SETUP' AND_FILE
  WORKFILE = SORTOUT
  return

CMPR_2_FIELDS_DIFF_FILES:
/*-------------------------------------------------------------------------*\
|  This routine processes a compare of two fields in two different files.    |
|  This could be a standard compare of a field in two input files or a       |
```

```
|  compare of a compound file (a previously combined file compare structure) |
|  to a generated SELECT output table file.  Either way, the displacements   |
|  to the fields are adjusted as necessary and a special routine processes   |
|  the comparison between the two files.                                     |
\*-------------------------------------------------------------------------*/
say '++++++++ COMPARE 2 FIELDS IN DIFFERENT FILES ++++++++'
  /*-- adjust for compound file needs --*/
  if COMPOUND = 'Y'    then
      if DDALIAS1 = DDALIAS.2    then
          AA = AA + DDLRECL.1
  /*-- sort first (compare) file into the needed order --*/
  drop SORTCARD.
  SORTCARD.0 = 1
  SORTCARD.1 = '  SORT FIELDS=('AA','BB',CH,A) '
  if AND_FILE = ''    then
      call SORT_DATA 'SETUP' DSN.DDALIAS1
  else
      call SORT_DATA 'SETUP' AND_FILE
  WORK1 = SORTOUT
  /*-- sort second (control) file into the needed order --*/
  drop SORTCARD.
  SORTCARD.1 = '  SORT FIELDS=('CC','DD',CH,A) '
  /*-- make the control file a DISTINCT list for IN and NI --*/
  if OPER = 'IN'  |  OPER = 'NI'    then
      do
        SORTCARD.0 = 2
        SORTCARD.2 = '  SUM FIELDS=NONE '
      end
  else
      do
        SORTCARD.0 = 1
        COMPOUND = 'Y'  /* set flag to indicate a compound output file */
      end
  call SORT_DATA 'SETUP' DSN.DDALIAS2
  WORK2 = SORTOUT
  /*-- compare the first and second files --*/
  call COMPARE_WORK1_WORK2
  WORKFILE = WORK3
  return
```

TEMPORARY OUTPUTTING d) – SORT DATA, used with the "PROCESS" option, dynamically runs SYNCSORT using the specified input file along with generated INREC control cards to do whatever task was specified. For specific purposes of this example, this iteration of the SORT_DATA paragraph selects data fields from selected combined-records temporary file. The resulting output is stored in the next designated temporary workfile defined by the SQL utility.

```
SORT_DATA:
/*-------------------------------------------------------------------------*\
|  Allocate input/output files needed and process the requested file SORT    |
|  Depending on SORT USE type, allocation for output is done differently     |
|    SETUP - indicates putting an input file into a required format and/or   |
|            order for actual processing.  This option essentially uses the  |
|            LRECL of the input file or the LRECL determined by SYNCSORT     |
|            due to use of an INREC or OUTREC option                         |
|    PROCESS - indicates actual processing of the file/s to select, reformat,|
|            merge, etc., to provide a requested function output.  This      |
|            option uses a predetermined LRECL for output to garantee that   |
|            all outputs of compares within WHERE logic are compatible for   |
|            later possible merge and/or combined SELECT reformatting.       |
\*-------------------------------------------------------------------------*/
  parse upper arg SORT_USE SORTIN
  /*---- processing messages ----*/
  SORT_CNT = SORT_CNT + 1
  SORTOUT = PNODE'.SQL.SORT.WORK'right(SORT_CNT,2,'0')
  WKDSN.0 = WKDSN.0 + 1
  interpret 'WKDSN.'WKDSN.0' = SORTOUT'    /* add work DSN to list */
  if SORTIN = '*MERGE*'    then
      do
        SORTIN = "'"MERG_LIST.1"'"
        do I1 = 2 to MERG_LIST.0
          SORTIN = SORTIN" '"MERG_LIST.I1"'"
        end
      end
  else
      SORTIN = "'"SORTIN"'"
  say ' '
  say ' * * *  SORT 'SORT_CNT' Diagnostics  * * *'
```

```
  say '     SORTIN  = 'SORTIN
  say '     SORTOUT = 'SORTOUT
  /*---- allocate input ----*/
 "alloc f(SORTIN) da("SORTIN") SHR"
  /*---- obtain file characteristics and allocate output file ----*/
  x = outtrap("DUMMY.")          /* trap err msgs... if any */
 "DELETE '"SORTOUT"'"            /* delete any old copy */
  x = outtrap("OFF")             /* turn trap back off */
  call SET_OUTPUT_UNITS_SPACE SORT_USE SORTIN
say '    + SORT_USE = 'SORT_USE'    + LRECL = 'LRECL
say '    + PRIMSPC = 'PRIMSPC'    + SECSPC = 'SECSPC'    + UNITS = 'UNITS
  if SORT_USE = 'SETUP'  |  LRECL = 0    then
      do
       "alloc f(SORTOUT) da('"SORTOUT"') new catalog release " ,
           " unit(SYSDA) space("PRIMSPC","SECSPC") "UNITS" " ,
           " dsorg(PS) recfm(F,B) blksize(0)"
      end
  else
      do
       "alloc f(SORTOUT) da('"SORTOUT"') new catalog release " ,
           " unit(SYSDA) space("PRIMSPC","SECSPC") "UNITS" " ,
           " dsorg(PS) recfm(F,B) lrecl("LRECL") blksize(0)"
      end
/* "alloc f(SORTWK01) unit(SYSDA) space(50,5) cylinders" */
/* "alloc f(SORTWK02) unit(SYSDA) space(50,5) cylinders" */
/* "alloc f(SORTWK03) unit(SYSDA) space(50,5) cylinders" */
 "alloc f(SYSIN) da(SYSIN) unit(SYSDA) space(1,1) tracks "  ,
     " dsorg(PS) recfm(F,B) lrecl(80) blksize(80) new delete"
 "execio * diskw SYSIN (stem SORTCARD. finis"
/* "alloc f(SYSOUT) DUMMY REUSE"  */
 "alloc f(SYSOUT) da(SYSOUT.S"time('S')") unit(SYSDA) space(1,1) TRACKS " ,
     " dsorg(PS) recfm(F,B,A) lrecl(133) blksize(0) new delete"
/* "call 'FDR.SYNCR36.LINKLIB(SYNCSORT)'"  */
  address ATTCHMVS "SORT"
  SORT_RC = RC
 "execio * diskr SYSOUT (stem SYSOUT. finis"
/* "free f(SORTIN SORTOUT SYSIN SYSOUT SORTWK01 SORTWK02 SORTWK03)" */
 "free f(SORTIN SORTOUT SYSIN SYSOUT)"
  do SSI = 1 to SYSOUT.0
    say SYSOUT.SSI
  end
  if SORT_RC > 0    then
      exit 16
  return
```

SELECTING DATA – c)

The following paragraph translates the user specified SELECT criteria into SYNCSORT INREC control cards the purpose of which are to extract specific data from a record of data and format the resulting output. Knowing that OUT#FLD.0 contains the number of output fields specified to be extracted (in this case 3… IN1.(5,45), IN2.(1,39),and IN2.(50,25)), this is accomplished by systematically stepping through the list of specified SELECT fields in the order they were requested and adding displacement/length information to the INREC control card/s as needed. SELECT data requested that resides in the tail end of the combined two-record information is addressed by using the respective SELECT displacement/length provided by the user and adding the LRECL (record length) of the leading record portion to the length.

```
GEN_INREC_CARD:
/*-------------------------------------------------------------------------*\
|    Generate an INREC FIELDS card to accomodate given SELECT fields        |
\*-------------------------------------------------------------------------*/
  LRECL = 0      /* NOTE: this setting of LRECL is also important for */
                 /*   later processing other than the following.      */

  if OUT#FLD.1 = '*'    then    /* default to copying the entire record */
      return

  INREC_CARD = '  INREC FIELDS=('
  do II = 1 to OUT#FLD.0
    if substr(OUT#FLD.II,1,1) = "'"    then
        do
          INREC_CARD = INREC_CARD'C'OUT#FLD.II','
          LRECL = LRECL + length(OUT#FLD.II) - 2
        end
    else
        do
          parse var OUT#FLD.II GICCHAR '.,' GICAA ',' GICBB
```

```
        if COMPOUND = 'Y'   then           /* if using compound file */
            if GICCHAR = DDALIAS.2   then  /* if field in 2nd part of file */
                GICAA = GICAA + DDLRECL.1  /* adjust the displacement */
        LRECL = LRECL + 1
        INREC_CARD = INREC_CARD||LRECL':'GICAA','GICBB','
        LRECL = LRECL + GICBB - 1
      end
    SORTCARD.0 = SORTCARD.0 + 1
    if II = OUT#FLD.0    then
        INREC_CARD = strip(INREC_CARD,t,',')')'
    interpret 'SORTCARD.'SORTCARD.0' = INREC_CARD'
    INREC_CARD = '                '
  end
  return

DISTINCT_CHECK:
/*-------------------------------------------------------------------------*\
|  Check to see if the DISTINCT option was requested.  If so generate the    |
|  necessary SORT FIELDS=(1,?,CH,A) and SUM FIELDS=NONE control cards...     |
|  otherwise generate a SORT FIELDS=COPY control card.                       |
\*-------------------------------------------------------------------------*/
  parse upper arg CHK_FILE
  if DISTINCT = ''    then    /* no DISTINCT option specified, so just copy */
      do                      /* the selected and/or reformatted records    */
        SORTCARD.0 = SORTCARD.0 + 1
        interpret "SORTCARD."SORTCARD.0" = '  SORT FIELDS=COPY '"
        return
      end

  if LRECL > 0    then    /* if an INREC card was generated prior... */
      do                  /* use the LRECL generated from that processing */
        SORTCARD.0 = SORTCARD.0 + 1
        interpret "SORTCARD."SORTCARD.0" = '  SORT FIELDS=(1,"LRECL",CH,A)'"
      end
  else         /* otherwise... */
      do
        /*  you probably got here because the distinct check was being done   */
        /*  for a SELECT or COUNT with a '*' (everything) designator.  You    */
        /*  didn't take care of that possibility yet... so fix it!            */
        say '***********  Probably processing a DISTINCT_CHECK for a SELECT or'
        say '** ERROR **  COUNT with a * field designation.  Did not program  '
        say '***********  for that one yet.  See the DV programmer to fix it. '
        exit 16
      end
  SORTCARD.0 = SORTCARD.0 + 1
  interpret "SORTCARD."SORTCARD.0" = '  SUM FIELDS=NONE'"
  return

COMPARE_WORK1_WORK2:
/*-------------------------------------------------------------------------*\
|  Compare work files WORK1 and WORK2 (WORK2 being the control file) using   |
|  the keys AA,BB and CC,DD (displacement and length) respectively.          |
|    If the SELECT verb did not specify a particular format, the default is  |
|  to select the output record from the compare file.                        |
\*-------------------------------------------------------------------------*/
  WORK_CNT = WORK_CNT + 1
  say ' '
  say ' * * *  COMPARE 'WORK_CNT' Diagnostics  * * *'
  /*---- prepare input files for use ----*/
  say '     WORK1 (Compare file) = 'WORK1
 "alloc f(WORK1) da('"WORK1"') shr"    /* alloc, */
 "execio 1 diskr WORK1 (stem WORK1."   /* open, and read the compare file */
  say '     WORK2 (Control file) = 'WORK2
 "alloc f(WORK2) da('"WORK2"') shr"    /* alloc, */
 "execio 1 diskr WORK2 (stem WORK2."   /* open, and read the control file */
  if OPER /= 'NE'    then
      call LOAD_CTRL_TBL /* deal with multiples of same key on control file */
  say '     SYSIN control cards'
  select
      when OPER = 'IN'    then
          say '           WORK1,'AA','BB' IN WORK2,'CC','DD
      when OPER = 'NI'    then
          say '           WORK1,'AA','BB' NOT-IN WORK2,'CC','DD
      when OPER = 'NE'    then
          say '           WORK1,'AA','BB' NOT= WORK2,'CC','DD
      otherwise
          say '           WORK1,'AA','BB' = WORK2,'CC','DD
  end
```

```
 /*---- prepare output file for use ----*/
 WORK3 = PNODE'.SQL.COMPARE.WORK'right(WORK_CNT,2,'0')  /* compare work DSN */
 WKDSN.0 = WKDSN.0 + 1
 interpret 'WKDSN.'WKDSN.0' = WORK3'        /* add compare work DSN to list */
 say '    WORK3 (Results file) = 'WORK3
 x = outtrap("DUMMY.")
"DELETE '"WORK3"'"                                  /* delete any old version */
 x = outtrap("OFF")
 /*---- set output units and amount of space needed ----*/
 if OPER = 'EQ'  |  OPER = 'NE'    then
     do
       UNITS = 'TRACKS'
       PRIMSPC = 450
       SECSPC = 150
     end
 else
     call SET_OUTPUT_UNITS_SPACE 'SETUP' "'"WORK1"'"
 /*---- allocate the output file ----*/
"alloc f(WORK3) da('"WORK3"') new catalog " ,
     "unit(SYSDA) "UNITS" space("PRIMSPC","SECSPC") release " ,
     "dsorg(PS) recfm(F,B) lrecl("WORK_LRECL") blksize(0)"
say '-- LRECL='WORK_LRECL
say '-- UNITS='UNITS
say '-- PRIMSPC='PRIMSPC
say '-- SECSPC='SECSPC
"execio 0 diskw WORK3 (open"  /* open output file for use */
 OUT_CNT = 0     /* initialize output record counter */
 if OPER = 'NI'    then /* processing for NOT IN disjoin */
     do forever
       select
         when substr(WORK1.1,AA,BB) = substr(CTRL.1,CC,DD)    then
           do
             call LOAD_CTRL_TBL                      /* load CTRL key data */
             WORK1_KEY = substr(WORK1.1,AA,BB)
            "execio 1 diskr WORK1 (stem WORK1."
             do while WORK1_KEY = substr(WORK1.1,AA,BB)  /* spin WORK1 file */
               if RC > 0    then leave                  /* data past the   */
              "execio 1 diskr WORK1 (stem WORK1."       /* current key     */
             end
             if RC > 0    then leave
           end
         when substr(WORK1.1,AA,BB) > substr(CTRL.1,CC,DD)    then
           call LOAD_CTRL_TBL
         otherwise
           do
             OUT_CNT = OUT_CNT + 1
             push WORK1.1
            "execio 1 diskw WORK3"
            "execio 1 diskr WORK1 (stem WORK1."
             if RC > 0    then leave
           end
       end
     end
 else
 if OPER = 'NE'    then   /* processing for NOT EQUAL disjoin */
     /*-- the NE compare form is essentially the Carsesian Product --*/
     /*-- of the two sets minus the equal keyed records ------------*/
     do while RC = 0
       /*-- write output for all keys < current compare record key --*/
       do while substr(WORK2.1,CC,DD) < substr(WORK1.1,AA,BB)  &  RC = 0
         push WORK1.1||WORK2.1
        "execio 1 diskw WORK3"
        "execio 1 diskr WORK2 (stem WORK2."
       end
       /*-- spin past equal keyed information --*/
       if RC = 0    then
           do while substr(WORK2.1,CC,DD) = substr(WORK1.1,AA,BB)  &  RC = 0
            "execio 1 diskr WORK2 (stem WORK2."
           end
       /*-- write output for all keys > current compare record key --*/
       if RC = 0    then
           do while RC = 0
             push WORK1.1||WORK2.1
            "execio 1 diskw WORK3"
            "execio 1 diskr WORK2 (stem WORK2."
           end
       /*-- close and reopen control file and read first record --*/
      "execio 0 diskr WORK2 (finis"
```

```
      "execio 1 diskr WORK2 (stem WORK2."
       /*-- read next compare file record --*/
      "execio 1 diskr WORK1 (stem WORK1."        /* current key     */
     end
  else    /* processing for IN join or a standard compare */
     do forever      /* actually, do until EOF on either file */
       select
```

SELECTING LIKE KEYED RECORDS - b)

The following eighteen statements are in play when identifying like keyed fields from records of two different files as in our simple example. This whole section of code under the COMPARE_WORK1_WORK2 (in red) paragraph name is specifically for comparing records from two different files as specified by the key field/s relational condition. For our example's purpose, the immediately following line selects like keyed records where substr(CTRL.1,CC,DD) is the key field substring (CC,DD being… starting in position 1 for a length of 4) of a record from file IN1 and substr(WORK1.1,AA,BB) is the key field substring (AA,BB being… starting in position 40 for a length of 4) of a record from file IN2. The following seventeen lines then write the resulting selected record/combined records to a temporary output file.

```
         when substr(WORK1.1,AA,BB) = substr(CTRL.1,CC,DD)    then
           do
             /*-- write a copy of the compare record for each control --*/
             /*-- record encountered with the same key ----------------*/
             OUT_CNT = OUT_CNT + CTRL.0  /* add to output count.*/
             if OPER = 'EQ'    then
                 do QQ = 1 to CTRL.0      /* for a multiple file compare, */
                   push WORK1.1||CTRL.QQ  /* combine the records for output */
                  "execio 1 diskw WORK3"  /* to aid in later possibilities */
                 end
             else
                 do QQ = 1 to CTRL.0      /* for an "IN" table compare, */
                   push WORK1.1           /* WORK1 is already in the format */
                  "execio 1 diskw WORK3"  /* required for output needs */
                 end
            "execio 1 diskr WORK1 (stem WORK1."
             if RC > 0    then leave
           end
         when substr(WORK1.1,AA,BB) > substr(CTRL.1,CC,DD)    then
           call LOAD_CTRL_TBL
         otherwise
           do
            "execio 1 diskr WORK1 (stem WORK1."
             if RC > 0    then leave
           end
       end
     end
 "execio 0 diskr WORK1 (finis"
 "execio 0 diskr WORK2 (finis"
 "execio 0 diskw WORK3 (finis"
 "free f(WORK1 WORK2 WORK3)"
  say '     Records Selected = 'OUT_CNT
  say ' '
  return

LOAD_CTRL_TBL:
/*-------------------------------------------------------------------------*\
|  To deal with possible mutiples of the same key in the control file, a    |
|  table of all control records with the current key is maintained.  Therefore|
|  each time there is a compare file match, a record is outputted for each  |
|  of the control files records with that key.                              |
|    NOTE:  This therefore allows for a possible cartesian product with the |
|  joining of two tables.                                                   |
\*-------------------------------------------------------------------------*/
  drop CTRL.
  QQ = 1
  CTRL.1 = WORK2.1  /* move current CTRL rec to the compare area */
  if substr(WORK2.1,CC,1) = 'Ÿ'    then nop /* if HV, prior read hit EOF */
  else
     do forever
      "execio 1 diskr WORK2 (stem WORK2."
       select
         when RC > 0    then     /* on EOF of the control file, */
             do                  /* move high-values into the record area */
               WORK2.0 = 1       /* so the compare file will always be less */
               WORK2.1 = overlay('Ÿ',WORK2.1,CC,DD,'Ÿ') /* than control key */
               leave
             end
```

```
          when substr(CTRL.1,CC,DD) = substr(WORK2.1,CC,DD)     then
              do
                QQ = QQ + 1
                CTRL.QQ = WORK2.1
/*----- test ----*/
if QQ//100 = 0     then
  do
    say '---- CTRL_TBL key = (WORK2.1,'CC','DD') = *'substr(WORK2.1,CC,DD)'*'
    say '    100 versions of the key encountered - you may wish to change how ',
        'the DVSQL request is stated to eliminate a cartesian product'
  end
/*----- test ----*/
              end
            otherwise leave
          end
        end
  CTRL.0 = QQ  /* set CTRL index to the number of recs with the current key */
  return

CLASS_COMPARE:
/*-------------------------------------------------------------------------*\
|  Generate the SORT INCLUDE control cards needed to accomplish the indicated |
|  data CLASS comparison.                                                     |
\*-------------------------------------------------------------------------*/
  OPER = substr(OPER,2)  /* drop the leading # class test indicator */
  select
    when OPER = 'ALPHA'  |  OPER = 'ALPHABETIC'     then
        do
          CC = left('A',BB,'A')
          DD = left('Z',BB,'Z')
          SORTCARD.1 = "  INCLUDE COND=("AA","BB",CH,GE,C'"CC"',AND,"
          SORTCARD.2 = "                "AA","BB",CH,LE,C'"DD"')"
          SORTCARD.0 = 2
        end
    when OPER = 'INTEGER'     then
        do
          CC = right('0',BB,'0')
          DD = left('9',BB,'9')
          SORTCARD.1 = "  INCLUDE COND=("AA","BB",CH,GE,C'"CC"',AND,"
          SORTCARD.2 = "                "AA","BB",CH,LE,C'"DD"')"
          SORTCARD.0 = 2
        end
    when OPER = 'ALPHANUMERIC'     then
        do
          CC = left('A',BB,'A')
          DD = left('Z',BB,'Z')
          SORTCARD.1 = "  INCLUDE COND=("AA","BB",CH,GE,C'"CC"',AND,"
          SORTCARD.2 = "                "AA","BB",CH,LE,C'"DD"',OR,"
          CC = right('0',BB,'0')
          DD = left('9',BB,'9')
          SORTCARD.3 = "                "AA","BB",CH,GE,C'"CC"',AND,"
          SORTCARD.4 = "                "AA","BB",CH,LE,C'"DD"')"
          SORTCARD.0 = 4
        end
    when OPER = 'BETWEEN'     then
        do
          /*-- get low value --*/
          if substr(WHERE_DATA,1,1) = "'"     then     /* literal */
              do
                CPOS = pos("'",WHERE_DATA,2) /* extract literal value... */
                FLD2 = substr(WHERE_DATA,1,CPOS)  /* might be embedded blanks */
                WHERE_DATA = strip(substr(WHERE_DATA,CPOS+1),'l')
              end
          else
              do
                say '**ERROR** BETWEEN format error'
                say '    low/first value not in quotes'
                say "    format is: BETWEEN 'low-value' AND 'high-value'"
                exit 16
              end
          /*-- extract AND --*/
          parse var WHERE_DATA JUNK WHERE_DATA
          if JUNK = 'AND'     then nop
          else
              do
                say '**ERROR** BETWEEN format error'
                say '    an AND did not follow the low/first value'
                say "    format is: BETWEEN 'low-value' AND 'high-value'"
```

```
            exit 16
          end
        /*-- get high value --*/
        if substr(WHERE_DATA,1,1) = "'"    then    /* literal */
          do
            CPOS = pos("'",WHERE_DATA,2) /* extract literal value... */
            FLD3 = substr(WHERE_DATA,1,CPOS)  /* might be embedded blanks */
            WHERE_DATA = strip(substr(WHERE_DATA,CPOS+1),'l')
          end
        else
          do
            say '**ERROR** BETWEEN format error'
            say '    high/second value not in quotes'
            say "    format is: BETWEEN 'low-value' AND 'high-value'"
            exit 16
          end
        /*-- check that first value is less than second value --*/
        if FLD2 < FLD3    then nop
        else
          do
            say '**ERROR** BETWEEN format error'
            say '    low/first value must be less than high/second value'
            say "    format is: BETWEEN 'low-value' AND 'high-value'"
            exit 16
          end
        /*-- build SORT cards --*/
        SORTCARD.1 = "  INCLUDE COND=("AA","BB",CH,GE,C"FLD2",AND,"
        SORTCARD.2 = "                "AA","BB",CH,LE,C"FLD3")"
        SORTCARD.0 = 2
      end
    otherwise nop
  end
  return

PROCESS_A_FUNCTION:
/*-------------------------------------------------------------------------*\
|   Do processing needed to obtain the result of a requested MIN, MAX, AVG,  |
|   SUM, COUNT, etc. function for the provided data set.                     |
|   The resulting value from processing the finction request is returned in  |
|   the result register/variable.                                            |
\*-------------------------------------------------------------------------*/
  parse arg FUNCTION
  say '    Processing the function: 'FUNCTION
  F_RESULT = ' '
 "alloc f(WORKFILE) da('"WORKFILE"') shr"
 "execio 0 diskr WORKFILE (open"
  select
    when FUNCTION = 'MAX'    then
      do
       "execio 1 diskr WORKFILE (stem WORKREC."
        if RC = 0    then
          F_RESULT = WORKREC.1
        do while RC = 0
          if WORKREC.1 > F_RESULT    then
            F_RESULT = WORKREC.1
         "execio 1 diskr WORKFILE (stem WORKREC."
        end
      end
    when FUNCTION = 'MIN'    then
      do
       "execio 1 diskr WORKFILE (stem WORKREC."
        if RC = 0    then
          F_RESULT = WORKREC.1
        do while RC = 0
          if WORKREC.1 < F_RESULT    then
            F_RESULT = WORKREC.1
         "execio 1 diskr WORKFILE (stem WORKREC."
        end
      end
    otherwise    /* default to COUNT */
      do
        F_RESULT = 0
       "execio 1 diskr WORKFILE (stem WORKREC."
        do while RC = 0
          F_RESULT = F_RESULT + 1
         "execio 1 diskr WORKFILE (stem WORKREC."
        end
      end
```

```
  end
 "execio 0 diskr WORKFILE (finis"
 "free f(WORKFILE)"
  say '        Result = 'F_RESULT
  return F_RESULT

SET_OUTPUT_UNITS_SPACE:
/*--------------------------------------------------------------------------*\
|    A generic routine used to determine output file UNITS and SPACE         |
|    allocation characteristics from listdsi information of the input file/s.|
|    Depending on ALLOCation USE, the variables are set differently:         |
|      SETUP - Indicates the file being generated is probably a copy or      |
|    reordered version of the original and needs to have the full volume of  |
|    space allocated to it.                                                  |
|      PROCESS - Indicates the file being generated is probably output of the|
|    actual SELECT reorder/reformat process and will most likely be          |
|    considerably smaller than its input version.                            |
\*--------------------------------------------------------------------------*/
  parse arg ALLOC_USE ALLOC_LIST
  if ALLOC_USE = 'SETUP'    then  /* alloc to contain same space as input */
      if pos(' ',ALLOC_LIST) > 0    then /* determine for multi-file input --*/
          do                              /* list that is to be merged -------*/
            x = listdsi("'"MERG_LIST.1"'")
            call SET_UNIT_TYPE
            PRIMSPC = SYSUSED
            SECSPC = SYSUSED
            do I1 = 2 to MERG_LIST.0
              x = listdsi("'"MERG_LIST.I1"'")
              PRIMSPC = PRIMSPC + SYSUSED
            end
          end
      else   /* use 50% of current used DASD for primary allocation and */
          do /* 25% of used DASD the secondary allocation */
            x = listdsi(ALLOC_LIST)
            if SYSREASON = 0    then
                do
                  call SET_UNIT_TYPE
                  PRIMSPC = format(SYSUSED*.5+1,,0)
                  SECSPC = format(SYSUSED*.25+1,,0)
                end
            else    /* probably not a PS or PO DSORG */
                do  /* use default values */
                  UNITS = 'TRACKS'
                  PRIMSPC = 300
                  SECSPC = 90
                end
          end
  else    /* allocate for SELECT processing... use LRECL determined by the */
      do  /* SELECT fields to provide a probable percentage of original    */
        x = listdsi(ALLOC_LIST)  /* input file volume */
        if SYSREASON = 0    then
            do
              call SET_UNIT_TYPE
say '   SYSUSED='SYSUSED'    LRECL='LRECL'    SYSLRECL='SYSLRECL
              PRIMSPC = format(SYSUSED*LRECL/SYSLRECL*.5+1,,0)
              SECSPC = format(SYSUSED*LRECL/SYSLRECL*.25+1,,0)
            end
        else    /* probably not a PS or PO DSORG */
            do  /* use default values */
              UNITS = 'TRACKS'
              PRIMSPC = 300
              SECSPC = 90
            end
      end
  /*-- set a minimum default space allocation just in case the input --*/
  /*-- file/s was empty ------------------------------------------------*/
  if PRIMSPC = 0    then
      do
        PRIMSPC = 1
        SECSPC = 1
      end
  return

SET_UNIT_TYPE:
  select
      when SYSUNITS = 'BLOCK'    then
          UNITS = SYSUNITS'('SYSBLKSIZE')'
      when SYSUNITS = 'TRACK'    then
```

```
        UNITS = 'TRACKS'
      when SYSUNITS = 'CYLINDER' then
        UNITS = 'CYLINDERS'
      otherwise
        UNITS = 'TRACKS'
  end
  return

IDCAMS_RENAME:
/*-------------------------------------------------------------------*\
|    Run a IDCAMS to rename a WORKFILE output to a user specified    |
|    final output DSName.                                            |
\*-------------------------------------------------------------------*/
  say ' '
  say ' '
  say '>>>> Output is on 'OUTDSN' <<<<'
  x = outtrap('DUMMY.')
 "delete '"OUTDSN"'"       /* delete any prior versions of output */
  x = outtrap('OFF')
  drop SYSIN.
  SYSIN.0 = 2
  SYSIN.1 = '  ALTER 'WORKFILE' - '
  SYSIN.2 = '          NEWNAME('OUTDSN')  '
 "alloc f(SYSIN) da(SYSIN) new delete unit(SYSDA) tracks space(1,1) ",
      "dsorg(PS) recfm(F,B) lrecl(80) blksize(0)"
 "execio * diskw SYSIN (stem SYSIN. finis"     /* put ctlcards on SYSIN */
/* "alloc f(SYSPRINT) DUMMY REUSE" */
 "alloc f(SYSPRINT) da(SYSPRINT) new delete unit(SYSDA) tracks space(1,1) ",
      "dsorg(PS) recfm(F,B) lrecl(133) blksize(0)"
 "call 'SYS1.LINKLIB(IDCAMS)'"
  IDCAMS_RC = RC
 "execio * diskr SYSPRINT (stem SYSPRT. finis"
 "free f(SYSIN SYSPRINT)"
  if IDCAMS_RC > 0     then
      do
        say '    **ERROR** encountered renaming workfile to user specified',
            'DSN 'OUTDSN
        do ISYSPRT = 1 to SYSPRT.0
          say '         'SYSPRT.ISYSPRT
        end
      end
  return

DELETE_WORK_DATA_SETS:
/*-------------------------------------------------------------------*\
|   Delete all of the intermediate work and compare data sets        |
|   generated throught SQL processing                                |
\*-------------------------------------------------------------------*/
  if WKDISP = 'KEEP'     then
      do
        say '---- Per user specified parm, intermediate work data sets',
            'will not be deleted'
        say "---- Look under '"PNODE".SQL.*' for the work, sort, and compare",
            "data sets used"
        return
      end
  else
      do
        say ' '
        say '---- Per user specified parm, intermediate SQL work data sets',
            'will be deleted ----'
      end
  x = outtrap('DUMMY.')
  if WKDSN.0 > 0     then
      do I = 1 to WKDSN.0
       "delete '"WKDSN.I"'"
        if RC = 0     then
            say '     'WKDSN.I'  -  deleted'
        else
            say '     problems deleting work data set 'WKDSN.I
      end
  x = outtrap('OFF')
  return

DETERMINE_LRECL:
/*-------------------------------------------------------------------*\
|    Determine LRECL from input file                                 |
\*-------------------------------------------------------------------*/
```

```
 parse var LRECL_DSN
 LRECL = 0
 x = listdsi("'"LRECL_DSN"'")
 if SYSREASON = 0    then
     LRECL = SYSLRECL
 else
 if SYSREASON = 12    then    /* VSAM input file */
     do
      "alloc f(SORTIN) da('"LRECL_DSN"') SHR"
       LRECLCHK = PNODE".LRECLCHK.S"time('S')
       x = outtrap("DUMMY.")
      "DELETE '"LRECLCHK"'" /* cleanup up possible prior version */
       x = outtrap("OFF")
      "alloc f(SORTOUT) da('"LRECLCHK"') new delete " ,
           " unit(SYSDA) space(1,1) tracks " ,
           " dsorg(PS) recfm(F,B) blksize(0)"
      "alloc f(SYSIN) da(SYSIN) unit(SYSDA) space(1,1) tracks "  ,
           " dsorg(PS) recfm(F,B) lrecl(80) blksize(80) new delete"
       LCHK.0 = 1
       LCHK.1 = '  SORT FIELDS=COPY,STOPAFT=1 '
      "execio 1 diskw SYSIN (stem LCHK. finis"
      "alloc f(SYSOUT) DUMMY"
       /* "call 'FDR.SYNCR36.LINKLIB(SYNCSORT)'"  */
       address ATTCHMVS "SORT"
       /* use LRECL of first record */
       if RC = 0    then
           do
             x = listdsi("'"LRECLCHK"'")
             LRECL = SYSLRECL
           end
       else
           say '**WARNING** could not obtain LRECL info for ',
               'the data set 'LRECL_DSN
      "free f(SORTIN SORTOUT SYSIN SYSOUT)"
     end
 else
     do
       say '**ERROR** Problem obtaining LRECL info for 'LRECL_DSN
       say '    Cannot process without it.  Processing terminated.'
       exit
     end
 return

/********************** C H A N G E   L O G ********************************/
/**  DATE      PGMR                 DESCRIPTION                            */
/*-------------------------------------------------------------------------*/
/*  [redacted]  lcframe  NOT IN added to WHERE verb options.                */
/*  [redacted]  lcframe  Cleanup of code, use full DD alias'.' in WHERE_TYPE.*/
/*  [redacted]  lcframe  Addition of WHERE IN (table list, etc.) option.     */
/*  [redacted]  lcframe  Upgrade product to handle VSAM and TAPE inputs and be */
/*     generally more generic in processing.                                */
/*  [redacted]  lcframe  Genericise generation of INREC and OUTREC ctlcards. */
/*  [redacted]  lcframe  Add option to specify by final output data set by   */
/*     input parm (using internal processing defaults) or specific INTO DD. */
/*  [redacted]  lcframe  Fix a problem with the unit specification associated */
/*     with the compare file parameters.                                    */
/*  [redacted]  lcframe  Complete adding the work data set DELETE/KEEP option */
/*  [redacted]  lcframe  Invoke SYNCSORT via ATTCHMVS to be able to more     */
/*     generally locate it on other JES complexes.                          */
/*  [redacted]  lcframe  Make sure the output file is at least empty.  No null */
/*     outputs allowed.                                                     */
/* 06/06/01  lcframe  Upgrade changes to allow for use of MIN, MAX, and     */
/*     COUNT functions as well as AND/OR logic in the WHERE verb.           */
/* 2004/01/15  lcframe  SORTWKxx files are no longer needed. SYNCSORT now   */
/*     dynamically monitors and allocates SORTWKxxs as needed.              */
/*************************************************************************/
```